Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

TREBALL FINAL DE MÀSTER

Numerical Simulation of Stationary Diffusion Problems using High-Order Hybridizable Continuous and Discontinuous Galerkin Methods

Treball realitzat per:
David Codony Gisbert

Dirigit per:
J. Sarrate, E. Ruiz-Gironés

Màster en:
Enginyeria de Camins, Canals i Ports

Barcelona, juny de 2016

Departament d'Enginyeria Civil i Ambiental

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Numerical Simulation of Stationary Diffusion Problems using High-Order Hybridizable Continuous and Discontinuous Galerkin Methods

by

Codony Gisbert, David

A thesis submitted in complete fulfillment for the
Master's Degree of Civil Engineering

in the

ETSECCPB
Escola de Camins

June 21, 2016

Declaration of Authorship

I, *David Codony Gisbert*, declare that this thesis titled *"Numerical Simulation of Stationary Diffusion Problems using High-Order Hybridizable Continuous and Discontinuous Galerkin Methods"* and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for the master's degree in civil engineering at Universitat Politècnica de Catalunya.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Acknowledgements

I would first like to thank my thesis advisors Dr. Josep Sarrate and Dr. Eloi Ruiz-Gironés of the *Laboratori de Càlcul Numèric* (LaCàN) at the *Universitat Politècnica de Catalunya* (UPC). They both convinced me to join this project, which I have really enjoyed throughout this year. They provided me with full support, establishing weekly meetings and maintaining continuous communication with me. Having them on my side, I have been able to discover a new horizon in terms of numerical methods and computational engineering. Moreover, I have learned about efficient programming techniques, specially by means of `Python`. Thanks to them I have discovered as well what I really like and which steps to follow in the near future.

I would also like to acknowledge the people I met during my internship at CIMNE, specially Rubén, Tomás, Flo and Vicente. They have been continuously interested in my academic progress and they supported me in the challenge of learning LaTeX and using it for the first time in my life in order to write this document.

I am also grateful to my friends for all the support they provided me during my academic life. Luis, Laura, Bret, Solà and Irene have been by my side in classroom, in the exams and during the sessions of late night study. We shared a lot of experiences both inside and outside of the university, and they have contributed to define who I currently am. I hope they will keep on contributing to it from now on. I also want to mention Ino and Dani, who have played an important role in my academic life, as well as out of it.

I want to dedicate some lines specifically to Laura. She already knows how much I appreciate her continuous interest in my thesis and her unfailing support in all aspects of my life.

Finally, I must express my gratitude to Carmen, Josep Maria, Sandra and Jordi for providing me with support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. I know they made a big effort to try to understand the basics of my research topics when I recently explained it to them. I appreciate it so much.

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Abstract

ETSECCPB
Escola de Camins

Master's Minor Thesis

by Codony Gisbert, David

The objective of this work is to analyze the advantages of several high-order finite element formulations to solve 2D and 3D stationary diffusion problems. This work states the first step of a long-term research project carried out by my mentors, which aims to develop a high-order finite element code able to simulate fluid flow through porous media in oilfields.

Both continuous and discontinuous element-wise polynomial methods are considered in this thesis. The former is the traditional Galerkin finite element approach. The latter provides the solution for both the scalar and the flux unknowns using a mixed formulation, that allows the local post-processing of the scalar unknown in order to increase its convergence rate. The use of high-order meshes instead of linear ones leads to an exponential error convergence, a higher accuracy level and to potentially faster computations if hybridization is considered. To this end, *four* codes are developed, considering continuous (CG) and discontinuous (DG) Galerkin methods, as well as their hybridized versions (HCG and HDG respectively). The implementation is carried out by means of `Python`.

The code is extensively validated in quadrilateral, triangular and hexahedral high-order meshes, in complex domains with curved boundaries and in heterogeneous domains. The efficiency of the code is assessed in terms of computing times and memory requirements. The benefits of the hybridization technique and high-order methods are highlighted. Continuous and discontinuous approaches are finally compared in these terms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Main Objectives

The objective of this thesis is to analyze high-order finite element formulations to solve stationary diffusion problems, in terms of accuracy and computational resources. In particular, we focus on the 2D and 3D linear Poisson equation. To this end, we have developed four high-order finite element codes, considering different formulations:

- The Continuous Galerkin method (CG)
- The hybridized version of the Continuous Galerkin method (HCG)
- The Discontinuous Galerkin method (DG)
- The hybridized version of the Discontinuous Galerkin method (HDG)

These codes have been efficiently constructed and rigorously validated, in order to allow us to make comparisons between continuous and discontinuous approaches, to assess the usefulness of the hybridization technique and to analyze the performance of high-order meshing.

1.2 Context of the Thesis

This work states the first step of a long-term research project carried out by my mentors, which aims to develop a high-order finite element code able to simulate fluid flow through porous media in oilfields. The code will be able to deal with multiple materials

interacting with each other, including changes of phase due to thermal and mechanical conditions over time.

The simulation of elliptic problems prior to considering the whole intended physical phenomena is a simplification which is useful to gain knowledge about high-order meshing, about the hybridization technique and about the advantages, drawbacks and limitations of the discontinuous formulation with respect to the continuous one.

1.3 Brief overview of Hybridizable Discontinuous Galerkin methods

Discontinuous Galerkin (DG) methods are widely known in the scientific community. The first DG method was developed by Reed and Hill [24] in 1973 for hyperbolic equations, and since then, several discontinuous schemes have been proposed by different authors, such as the *interior penalty* (IP) family [9] in 1976. Discontinuous methods have been extended to elliptic and parabolic equations since then.

DG methods have been combined with mixed formulations [2] too, e.g. with the mixed method of Raviart-Thomas (RT) for symmetric second-order elliptic problems [23] or with the Brezzi-Douglas-Marini (BDM) mixed method [3]. DG mixed schemes continued being an interesting research field with the *local discontinuous Galerkin* (LDG) family [5], first developed in 1988. Mixed methods are characterized by considering as unknowns both the scalar u and the flux $\boldsymbol{q}$. An interesting property of the mixed methods is that both the scalar unknown and the flux converge at a rate of $(p+1)$ in the L_2-norm when the shape functions used consist on polynomials of order p [20]. For DG mixed methods with $p \geq 1$, the convergence rate of the scalar solution can be increased up to $(p+2)$ in the L_2-norm performing a simple *element by element* post-processing procedure (see section 3.2.4).

In 2004, [4] considered the hybridization of mixed methods. The distinctive feature of *hybridizable* Discontinuous Galerkin (HDG) methods is that the only globally coupled degrees of freedom are those of an approximation of the solution defined only on the boundaries of the elements [7], the numerical traces. Note then that not all mixed DG schemes can be hybridized.

In this thesis we implement both continuous and discontinuous approaches using high-order meshes, which has been shown to be efficient in hybrid schemes, since the addition of nodal unknowns is compensated by the condensation of the inner ones.

The continuous approach, which is *per se* hybridizable, has been implemented in its traditional (CG) and hybridized (HCG) versions, this latter using the standard static condensation technique.

The discontinuous approach is based on the so-called Local Discontinuous Galerkin - Hybridizable (or simply LDG-H) method (see [7, 15] for details), and similarly to the continuous approach, both its non-hybridized (DG) and hybridized (HDG) versions have been implemented.

1.4 Structure of the Document

This thesis is organized as follows. Chapter 2 presents the continuous and discontinuous formulations for the Poisson equation. In addition, we explain the static condensation procedure, known as hybridization in the scope of continuous and discontinuous schemes. In chapter 3 we focus on the implementation details of these formulations. Chapter 4 is devoted to illustrate the performance of the four implemented algorithms. In particular we discuss their capabilities and drawbacks. Finally, the thesis ends with a conclusion (chapter 5), which summarizes all the work done and presents the future work.

Chapter 2

Formulation

The purpose of this chapter is to detail the formulations of the different Galerkin formulations presented in this thesis in the case of two- and three-dimensional elliptic operators.
Section 2.1 explains the traditional Continuous Galerkin (*CG*) scheme. Static condensation procedure is explained in section 2.2 and used in the previous CG scheme in order to get the *statically condensed* or *hybridized* version of it. For the sake of simplicity we will name it as *HCG* in the scope of this thesis. Section 2.3 introduces the family of Discontinuous Galerkin methods using a mixed formulation for both scalar and flux unknowns. In particular, the LDG-H (local discontinuous Galerkin - hybridizable) method is implemented, which is hybridizable and can be locally post-processed in order to gain accuracy and improve convergence results once the global system has been solved. This section shows its non-hybridized version, which will be named as *DG* all along this thesis. Finally, section 2.4 shows the hybrid approach of the previous LDG-H scheme, called the *HDG* method. This latter formulation is the one that motivates the whole study.

2.1 CG Formulation

The well-known Continuous Galerkin (CG) method is next presented. The strong form of the elliptic diffusion problem is:

$$\nabla\cdot(\mathbf{D}(\boldsymbol{x})\cdot\nabla u(\boldsymbol{x})) = -f(\boldsymbol{x}), \qquad \boldsymbol{x}\in\Omega; \tag{2.1a}$$

$$u(\boldsymbol{x}) = g_D(\boldsymbol{x}), \qquad \boldsymbol{x}\in\partial\Omega_D; \tag{2.1b}$$

$$-\mathbf{D}(\boldsymbol{x})\cdot\nabla u(\boldsymbol{x})\cdot\boldsymbol{n} = g_N(\boldsymbol{x}), \qquad \boldsymbol{x}\in\partial\Omega_N; \tag{2.1c}$$

where Ω represents the whole domain, $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$ represents its boundary, which is formed by the Dirichlet boundary, $\partial\Omega_D$, and the Neumann boundary, $\partial\Omega_N$, verifying that $\partial\Omega_D \cap \partial\Omega_N = \emptyset$. The scalar $u(\boldsymbol{x})$ is the *problem unknown*. It can refer to the temperature in the thermal diffusive problem, or to the hydraulic head in the problem of fluid flow through porous media, to name a few particularizations of the Poisson problem. $\mathbf{D}(\boldsymbol{x})$ represents the *diffusion tensor*, which is given by a symmetric positive definite matrix. It defines in which way the diffusion process takes place. It refers to the thermal conductivity in the thermal problem, and to the soil permeability in the fluid flow through porous media problem. Finally, $f(\boldsymbol{x})$ is the *source term* of Ω, and $\boldsymbol{n}$ refers to the external unit normal of $\partial\Omega$. Equations (2.1b) and (2.1c) hold Dirichlet and Neumann boundary conditions for this problem, respectively.

2.1.1 Weak Form

Let us define the space of test functions V_0 as:

$$V_0 = \left\{ v \in L^2(\Omega) \quad | \quad v = 0 \text{ at } \partial\Omega_D \right\}. \tag{2.2}$$

The weak form derivation of this problem begins multiplying eq. (2.1a) by any possible test function $v \in V_0$ and integrating in the whole domain:

$$\int_\Omega v \nabla \cdot (\mathbf{D} \cdot \nabla u) \, \mathrm{d}\boldsymbol{x} = -\int_\Omega v f \, \mathrm{d}\boldsymbol{x}, \qquad \forall v \in V_0. \tag{2.3}$$

Next step is to integrate by parts the left hand side of eq. (2.3) and to apply then the divergence theorem. This leads to

$$-\int_\Omega \nabla v \cdot \mathbf{D} \cdot \nabla u \, \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} v \cdot (\mathbf{D} \cdot \nabla u) \cdot \boldsymbol{n} \, \mathrm{d}s = -\int_\Omega v f \, \mathrm{d}\boldsymbol{x}, \qquad \forall v \in V_0. \tag{2.4}$$

The second term on the left hand side of eq. (2.4) vanishes along the Dirichlet boundary, since the test function v does so; consequently, to integrate along the full boundary is equivalent as to integrate just along the Neumann boundary. The Neumann boundary condition shown in eq. (2.1c) is then explicitly imposed in eq. (2.4).

The weak form of the continuous Galerkin problem is then:

$$\begin{aligned} &\textit{Find } u \in U_D = \left\{ u \in L^2(\Omega) \quad | \quad u = g_D \textit{ at } \partial\Omega_D \right\} \textit{ such that} \\ &\int_\Omega \nabla v \cdot \mathbf{D} \cdot \nabla u \, \mathrm{d}\boldsymbol{x} = \int_\Omega v f \, \mathrm{d}\boldsymbol{x} - \int_{\partial\Omega_N} v g_N \, \mathrm{d}s, \qquad \forall v \in V_0. \end{aligned} \tag{2.5}$$

The Dirichlet boundary condition in (2.1b) is imposed in the set of admissible functions U_D.

2.1.2 Discretization

The domain Ω can be discretized in several smaller elements, which conform the mesh of the domain. This mesh is formed by N nodes. The scalar unknown $u(\boldsymbol{x})$ can be approximated in terms of a linear combination of simpler functions $\phi_j(\boldsymbol{x})$ (the *shape functions*) as

$$u(\boldsymbol{x}) \approx \sum_{j=1}^{N} \phi_j(\boldsymbol{x}) a_j. \tag{2.6}$$

The unknown is not $u(\boldsymbol{x})$ anymore, but the vector $\boldsymbol{a}$ ($\boldsymbol{a}[j] = a_j$, $\forall j = 1 \dots N$).

The $\phi_j(\boldsymbol{x})$ shape functions are defined continuous element-wise polynomials of degree p, in such a way that $\phi_i(\boldsymbol{x}_j) = \delta_{ij}$. From this property it comes out that the coefficient a_j represents the value of the scalar unknown u at the node $\boldsymbol{x}_j$, or equivalently $a_j = u(\boldsymbol{x}_j)$. We can sepparate the nodes of the mesh then in two sets: *a*) the N_{unk} nodes that do not belong to the Dirichlet boundary of Ω ($\boldsymbol{x}_j \notin \partial\Omega_D$) and *b*) the $N - N_{unk}$ nodes that belong to the Dirichlet boundary of Ω ($\boldsymbol{x}_k \in \partial\Omega_D$). Thus, the Dirichlet condition in eq. (2.1b) can be imposed in equation (2.6) because $a_k = u(\boldsymbol{x}_k) = g_D(\boldsymbol{x}_k)$ in the Dirichlet boundary. Then, we get:

$$u(\boldsymbol{x}) \approx \sum_{j=1}^{N_{unk}} \phi_j(\boldsymbol{x}) a_j + \sum_{k=N_{unk}+1}^{N} \phi_k(\boldsymbol{x}) g_D(\boldsymbol{x}_k). \tag{2.7}$$

Note that the first term on the right hand side of eq. (2.7) is now a sum only for the N_{unk} nodes which do not belong to the Dirichlet boundary, and not for all the N nodes of the mesh as before. The gradient of u is:

$$\nabla u(\boldsymbol{x}) \approx \sum_{j=1}^{N_{unk}} \nabla\phi_j(\boldsymbol{x}) a_j + \sum_{k=N_{unk}+1}^{N} \nabla\phi_k(\boldsymbol{x}) g_D(\boldsymbol{x}_k). \tag{2.8}$$

It still remains to determine the test functions v. Depending on the definition of v, we end up with one method or another. The Galerkin method is the one considered in this thesis, which uses the same shape functions $\phi_j(\boldsymbol{x})$ previously seen for the discretization of u in order to get the numerical solution for each element.

In order to approximate the space V_0 we choose V_0^h, the space of test functions, as:

$$V_0^h = \text{span } \{\phi_i(\boldsymbol{x}), \forall i = 1 \dots N_{unk}\}, \tag{2.9}$$

Thus, we set

$$v(\boldsymbol{x}) = \phi_i(\boldsymbol{x}), \qquad \forall i = 1 \dots N_{unk}, \tag{2.10}$$

and

$$\nabla v(\boldsymbol{x}) = \nabla \phi_i(\boldsymbol{x}), \qquad \forall i = 1 \dots N_{unk}. \tag{2.11}$$

We can insert equations (2.7), (2.8) and (2.11) in the weak form in (2.5) to get

$$\int_\Omega \nabla \phi_i \cdot \mathbf{D} \cdot \left(\sum_{j=1}^{N_{unk}} \nabla \phi_j(\boldsymbol{x}) a_j + \sum_{k=N_{unk}+1}^{N} \nabla \phi_k(\boldsymbol{x}) g_D(\boldsymbol{x}_k) \right) \mathrm{d}\boldsymbol{x} = \int_\Omega \phi_i f \, \mathrm{d}\boldsymbol{x} - \int_{\partial\Omega_N} \phi_i g_N \, \mathrm{d}s, \quad \forall i = 1 \dots N_{unk}. \tag{2.12}$$

Since the left hand side of eq. (2.12) is bilinear and $g_D(\boldsymbol{x}_k)$ and a_j do not depend on the $\boldsymbol{x}$ coordinate, this term of the equation can be split in two parts (the known, which depends on $g_D(\boldsymbol{x}_k)$ and the unknown depending on a_j), and both $g_D(\boldsymbol{x}_k)$ and a_j can get out of the integral. The final expression of the weak form discretization of the traditional Continuous Galerkin elliptic problem is:

Find $a_1, \dots, a_{N_{unk}}$, *such that,* $\forall i = 1 \dots N_{unk}$,

$$\sum_{j=1}^{N_{unk}} \int_\Omega \nabla \phi_i \cdot \mathbf{D} \cdot \nabla \phi_j \, \mathrm{d}\boldsymbol{x} \cdot a_j = \int_\Omega \phi_i f \, \mathrm{d}\boldsymbol{x} - \int_{\partial\Omega_N} \phi_i g_N \, \mathrm{d}s - \sum_{k=N_{unk}+1}^{N} \int_\Omega \nabla \phi_i \cdot \mathbf{D} \cdot \nabla \phi_k \, \mathrm{d}\boldsymbol{x} \cdot g_D(\boldsymbol{x}_k). \tag{2.13}$$

The left hand side of eq. (2.13) refers to the *stiffness* of the system multiplied by the unknowns vector, $\boldsymbol{a}$. The right hand side of the same equation refers to the load vectors accounting for the source term, the Neumann condition and the Dirichlet condition. This equation can be written in matrix form according to the next definitions:

$$\mathbf{K}[i,j] := \int_\Omega \nabla \phi_i \cdot \mathbf{D} \cdot \nabla \phi_j \, \mathrm{d}\boldsymbol{x}, \qquad \forall i,j = 1 \dots N_{unk}, \tag{2.14a}$$

$$\boldsymbol{f}[i] := \boldsymbol{f}_f[i] - \boldsymbol{f}_N[i] - \boldsymbol{f}_D[i], \qquad \forall i = 1 \dots N_{unk}; \tag{2.14b}$$

where

$$\boldsymbol{f}_f[i] := \int_\Omega \phi_i f \, \mathrm{d}\boldsymbol{x}, \tag{2.15a}$$

$$\boldsymbol{f}_N[i] := \int_{\partial\Omega_N} \phi_i g_N \, \mathrm{d}s, \tag{2.15b}$$

$$\boldsymbol{f}_D[i] := \int_\Omega \nabla \phi_i \cdot \mathbf{D} \cdot \nabla \phi_k \, \mathrm{d}\boldsymbol{x} \cdot g_D(\boldsymbol{x}_k). \tag{2.15c}$$

Then, the matrix form of the Continuous Galerkin elliptic problem is

$$\mathbf{K} \cdot \boldsymbol{a} = \boldsymbol{f}. \tag{2.16}$$

The matrix $\mathbf{K}$ and the vector $\boldsymbol{f}$ are frequently computed in terms of its local contributions $\mathbf{K}^e$ and $\boldsymbol{f}^e$, which are assembled as usual:

$$\mathbf{K} = \bigwedge_{\Omega^e \in \Omega} \mathbf{K}^e, \qquad \boldsymbol{f} = \bigwedge_{\Omega^e \in \Omega} \boldsymbol{f}^e, \tag{2.17}$$

being $\bigwedge$ the assembly operator that assembles each elemental matrix in the global matrix. The vector $\boldsymbol{a}$ can be found out solving this algebraic linear system, and then the solution of the whole problem is retrieved using eq. (2.7).

2.2 CG Formulation with Static Condensation

2.2.1 Static Condensation

The static condensation technique is known as a specific process for reducing the number of unknowns of a global system of equations before solving it. It consists on splitting the degrees of freedom in two sets: *a*) the *primary unknowns*, which are the ones that will remain in the system, and *b*) the *secondary unknowns* to be condensed. This latter ones will be expressed in terms of the primary unknowns, in such a way that the system of equations will be reduced just to the primary unknowns. See [10, 21, 27, 30] for detailed information.

Let us consider as an example the linear system $\mathbf{A} \cdot \boldsymbol{x} = \boldsymbol{b}$. Let us label as 1 the primary set of unknowns, and the secondary one as 2 . Without loss of generality, we can reorder the equations according to its set label, leading to

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \cdot \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix}. \tag{2.18}$$

If the submatrix $\mathbf{A}_{22}$ in eq. (2.18) is invertible, then we can write

$$\boldsymbol{x}_2 = \mathbf{A}_{22}^{-1} \cdot \{\boldsymbol{b}_2 - \mathbf{A}_{21} \cdot \boldsymbol{x}_1\} \tag{2.19}$$

and insert eq. (2.19) in the first subset of eq. (2.18) to get

$$\mathbf{A}_{11} \cdot \boldsymbol{x}_1 + \mathbf{A}_{12} \cdot \{\mathbf{A}_{22}^{-1} \cdot \{\boldsymbol{b}_2 - \mathbf{A}_{21} \cdot \boldsymbol{x}_1\}\} = \boldsymbol{b}_1. \tag{2.20}$$

Reorganizing terms leads us to the standard Schur complement formulation of the statically condensed problem for the primary unknowns:

$$\{\mathbf{A}_{11} - \mathbf{A}_{12} \cdot \mathbf{A}_{22}^{-1} \cdot \mathbf{A}_{21}\} \cdot \boldsymbol{x}_1 = \boldsymbol{b}_1 - \mathbf{A}_{12} \cdot \mathbf{A}_{22}^{-1} \cdot \boldsymbol{b}_2. \tag{2.21}$$

Note that $\boldsymbol{x}_2$ does not appear anymore in eq. (2.21) since it has been *condensed*. Once equation 2.21 is solved, $\boldsymbol{x}_1$ is introduced in eq. (2.19) to compute $\boldsymbol{x}_2$.

This procedure reduces the size of the linear system, specially when $\dim(x_1) \ll \dim(x_2)$, and increases the computational efficiency, specially when $\mathbf{A}_{22}^{-1}$ is easy to compute.

2.2.2 Static Condensation application in CG Formulation

In order to optimize the implementation of the CG formulation presented in section 2.1, it is possible to apply the static condensation procedure to it. In this way, the linear system in equation (2.16) will decrease its size, leading to less memory requirements and shorter CPU-times to solve it.

The idea is to apply the procedure seen in section 2.2.1 to the equation (2.16). Instead of applying static condensation to the global linear system, we condensate the local contribution of each element before the assembly process. First, we define two sets: *a*) the set of the primary unknowns containing all nodes of element e that belong to the boundary of that element ($\boldsymbol{x}_j^e \in \partial\Omega^e$) and *b*) the set of the secondary unknowns containing the inner nodes of the element Ω^e ($\boldsymbol{x}_j^e \notin \partial\Omega^e$). We label those sets as b (for *boundary*) and i (for *inner*) respectively. The local approach is valid since the matrix of the inner nodes of the system, $\mathbf{K}_{ii}$, is block-diagonal in smaller submatrices, $\mathbf{K}_{ii}^e$. This is because the inner nodes within one element are not related with the inner nodes of another element. We write then the elemental contributions as

$$\mathbf{K}^e := \begin{pmatrix} \mathbf{K}_{bb}^e & \mathbf{K}_{bi}^e \\ \mathbf{K}_{ib}^e & \mathbf{K}_{ii}^e \end{pmatrix}, \qquad \boldsymbol{a}^e := \begin{bmatrix} \boldsymbol{a}_b^e \\ \boldsymbol{a}_i^e \end{bmatrix}, \qquad \boldsymbol{f}^e := \begin{bmatrix} \boldsymbol{f}_b^e \\ \boldsymbol{f}_i^e \end{bmatrix}. \tag{2.22}$$

For valid elements, the submatrix of the inner nodes $\mathbf{K}_{ii}^e$ is always non-singular, since it relates the inner nodes to an elliptic Dirichlet boundary problem at the element level (see details in [15]). Therefore, the static condensation technique seen in section 2.2.1 can be applied in the same way to get the hybridized elemental contributions

$$\mathbf{K}_H^e = \{\mathbf{K}_{bb}^e - \mathbf{K}_{bi}^e \cdot (\mathbf{K}_{ii}^e)^{-1} \cdot \mathbf{K}_{ib}^e\} \tag{2.23a}$$

$$\boldsymbol{f}_H^e = \boldsymbol{f}_b^e - \mathbf{K}_{bi}^e \cdot (\mathbf{K}_{ii}^e)^{-1} \cdot \boldsymbol{f}_i^e. \tag{2.23b}$$

Note that the size of $\mathbf{K}_{ii}^e$ is relatively small using this local approach (e.g. for a 2-D quadrilateral mesh, using polynomial shape functions of degree $p = 5$, the size of $\mathbf{K}_{ii}^e$ is only 4×4), so the computation of its inverse is affordable.

The assembly process of $\mathbf{K}_H$ and $\boldsymbol{f}_H$ for the whole system is performed in the following way:

$$\mathbf{K}_H = \bigwedge_{\Omega^e \in \Omega} \mathbf{K}_{bb}^e - \mathbf{K}_{bi}^e \cdot (\mathbf{K}_{ii}^e)^{-1} \cdot \mathbf{K}_{ib}^e, \qquad \boldsymbol{f}_H = \bigwedge_{\Omega^e \in \Omega} \boldsymbol{f}_b^e - \mathbf{K}_{bi}^e \cdot (\mathbf{K}_{ii}^e)^{-1} \cdot \boldsymbol{f}_i^e, \quad (2.24)$$

so the final form of the linear system for the condensed problem is

$$\mathbf{K}_H \cdot \boldsymbol{a}_H = \boldsymbol{f}_H. \quad (2.25)$$

The size of the linear system in (2.25) is $N_b \times N_b$, being N_b the number of nodes of the whole mesh that belong to the boundary of an element that do not belong to the Dirichlet boundary. So the inner nodes of the mesh have been then *condensed* and are not present in (2.25). The vector $\boldsymbol{a}_H$, which contains the values of the scalar unknown only for the nodes on the boundary of the elements that do not belong to the Dirichlet boundary, is obtained once the linear system in (2.25) is solved. The value of the scalar unknown at the inner nodes of the elements of the mesh can be computed *element by element* as

$$\boldsymbol{a}_i^e = (\mathbf{K}_{ii}^e)^{-1} \cdot \{\boldsymbol{f}_i^e - \mathbf{K}_{ib}^e \cdot \boldsymbol{a}_b^e\}. \quad (2.26)$$

This particular approach avoids the assembly of both the full matrix and the full load vector, as done in the CG case in section 2.1.2, and reduces the costs of the computation. Moreover, it allows the parallel computation of the inner nodes of each element once the scalar unknown along its boundaries has been computed.

2.3 DG Formulation

LDG-H method is defined using the mixed formulation, which introduces an auxiliary flux variable $\boldsymbol{q}$ in order to write two first-order differential equations (2.27a), (2.27b) for the traditional elliptic diffusion problem:

$$\nabla \cdot \boldsymbol{q}(\boldsymbol{x}) = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega; \quad (2.27a)$$

$$\mathbf{D}(\boldsymbol{x}) \cdot \nabla u(\boldsymbol{x}) + \boldsymbol{q}(\boldsymbol{x}) = \boldsymbol{0}, \qquad \boldsymbol{x} \in \Omega; \quad (2.27b)$$

$$u(\boldsymbol{x}) = g_D(\boldsymbol{x}), \qquad \boldsymbol{x} \in \partial\Omega_D; \quad (2.27c)$$

$$\boldsymbol{q}(\boldsymbol{x}) \cdot \boldsymbol{n} = g_N(\boldsymbol{x}), \qquad \boldsymbol{x} \in \partial\Omega_N. \quad (2.27d)$$

Note that the Dirichlet and Neumann boundary conditions appear in (2.27c) and (2.27d), respectively.

2.3.1 Weak Form

Equation (2.27b) can be modified in order to simplify the weak formulation and optimize the computation of the elemental contributions. Taking advantage of the properties of the permeability tensor $\mathbf{D}$ (which is symmetric and positive definite), it is possible to rewrite (2.27b) as

$$\nabla u(\boldsymbol{x}) + \mathbf{D}^{-1}(\boldsymbol{x}) \cdot \boldsymbol{q}(\boldsymbol{x}) = \boldsymbol{0}, \qquad \boldsymbol{x} \in \Omega, \tag{2.28}$$

since $\mathbf{D}$ will always be invertible. This procedure is also applied in [20] and [4], among others. The advantages of following this approach will be explained in next lines.

In a DG framework, the unknowns u and $\boldsymbol{q}$ are discontinuous between elements. For this reason the strong form shown in (2.27a) and (2.28) has to be fulfilled in a discontinuous manner between elements.

Let us define the test functions v and $\boldsymbol{w}$ for scalar and vector fields of Ω, respectively, and μ for scalar fields of $\partial\Omega^e$. The weak form derivation of this problem starts then with (2.29):

$$\sum_{\Omega^e \in \Omega} \int_{\Omega^e} v \nabla \cdot \boldsymbol{q} \, \mathrm{d}\boldsymbol{x} = \sum_{\Omega^e \in \Omega} \int_{\Omega^e} v f \, \mathrm{d}\boldsymbol{x}, \quad \forall v \in L^2(\Omega); \tag{2.29a}$$

$$\sum_{\Omega^e \in \Omega} \int_{\Omega^e} \boldsymbol{w} \cdot \mathbf{D}^{-1} \cdot \boldsymbol{q} \, \mathrm{d}\boldsymbol{x} + \sum_{\Omega^e \in \Omega} \int_{\Omega^e} \boldsymbol{w} \cdot (\nabla u) \, \mathrm{d}\boldsymbol{x} = 0, \qquad \forall \boldsymbol{w} \in [L^2(\Omega)]^{n_{sd}}. \tag{2.29b}$$

Next step is to apply integration by parts. Note that the permeability tensor appears only in the first term of equation (2.29b), which is not being integrated by parts. Using equation (2.28) instead of (2.27b) simplifies the weak form of the problem, since the integration by parts of its second term is performed as usual. Moreover, with this approach $\mathbf{D}$ is not required to be differentiable.

The integration by parts and the divergence theorem lead to the following set of equations:

$$\sum_{\Omega^e\in\Omega}\left(\int_{\partial\Omega^e} v\widetilde{\boldsymbol{q}}\cdot\boldsymbol{n}\,\mathrm{d}s-\int_{\Omega^e}(\nabla v)\cdot\boldsymbol{q}\,\mathrm{d}\boldsymbol{x}\right)=\sum_{\Omega^e\in\Omega}\int_{\Omega^e} vf\,\mathrm{d}\boldsymbol{x},\quad \forall v\in L^2(\Omega);\quad (2.30a)$$

$$\sum_{\Omega^e\in\Omega}\left(\int_{\Omega^e}\boldsymbol{w}\cdot\mathbf{D}^{-1}\cdot\boldsymbol{q}\,\mathrm{d}\boldsymbol{x}+\int_{\partial\Omega^e}(\boldsymbol{w}\cdot\boldsymbol{n})\widetilde{u}\,\mathrm{d}s - \sum_{\Omega^e\in\Omega}\int_{\Omega^e}(\nabla\cdot\boldsymbol{w})u\,\mathrm{d}\boldsymbol{x}\right)=0,\qquad \forall\boldsymbol{w}\in[L^2(\Omega)]^{n_{sd}},\quad (2.30b)$$

where u and $\boldsymbol{q}$ have been substituted at $\partial\Omega^e$ by their corresponding numerical traces $\widetilde{u}$ and $\widetilde{\boldsymbol{q}}$, which are defined on the boundary of each element. This procedure is done because neither u nor $\boldsymbol{q}$ are defined on $\partial\Omega^e$.

The traditional DG schemes are characterised precisely by the definition of their numerical traces. In our case, since the idea is to implement an *hybridizable* formulation, it comes up that the numerical traces are the *only* unknowns that can be coupled globally. This means that the scalar unknown u and the flux $\boldsymbol{q}$ within a particular element can only depend on themselves and on the numerical traces along the boundary of that element or, conversely, that a given numerical trace can only depend on information of their (two) adjacent elements and their surrounding numerical traces.

According to [7, 15], for any Ω^e element we can express the numerical trace of the flux $\widetilde{\boldsymbol{q}}^e$ in terms of u^e, $\boldsymbol{q}^e$, λ^e and the traces $\widetilde{u}^e$, usually denoted as λ^e, as:

$$\widetilde{\boldsymbol{q}}^e(\boldsymbol{x}):=\boldsymbol{q}^e(\boldsymbol{x})+\tau\cdot(u^e(\boldsymbol{x})-\lambda^e(\boldsymbol{x}))\cdot\boldsymbol{n}^e\qquad\text{on }\partial\Omega^e.\quad (2.31)$$

Equation (2.31) turns out to be the *only* possible definition of $\widetilde{\boldsymbol{q}}^e(\boldsymbol{x})$ if we require it to be (i) linearly dependent on the primitive u^e and the flux $\boldsymbol{q}^e$, (ii) consistent ($\widetilde{u}^e=u^e|_{\partial\Omega^e_D}=\lambda^e$, $\widetilde{\boldsymbol{q}}^e\cdot\boldsymbol{n}^e=(\boldsymbol{q}^e\cdot\boldsymbol{n}^e)|_{\partial\Omega^e_D}$) and (iii) its normal component to be single-valued; see detail in [1, 15]. This last property is equivalent as to say that the normal component of the numerical trace is *locally conservative*:

$$\int_{\partial S}\widetilde{\boldsymbol{q}}\cdot\boldsymbol{n}_S\,\mathrm{d}s+\int_S f\,\mathrm{d}\boldsymbol{x}=0,\quad (2.32)$$

being S the union of any collection of elements [1]. The penalty function usually considered by the local discontinuous Galerkin (LDG) schemes is set here as a factor τ, assumed to be positive so that the existence and uniqueness of the approximate solution is guaranteed [7, 15]. Other authors consider the possibility of this τ parameter being coordinate-dependent, but we will consider it as constant for the sake of simplicity.

Since we introduced a new unknown λ in the problem, we need a new equation to determine it. This equation is known as the *transmissivity condition*:

$$\sum_{\Omega^e \in \Omega} \int_{\partial\Omega^e} \mu \widetilde{\boldsymbol{q}} \cdot \boldsymbol{n} \, \mathrm{d}s = \int_{\partial\Omega_N} \mu g_N \, \mathrm{d}s, \qquad \forall \mu \in L^2(\Gamma \backslash \partial\Omega_D), \tag{2.33}$$

where $\Gamma = \cup_e \partial\Omega^e$. The transmissivity condition forces the normal component of the numerical flux to coincide from one element to the adjacent one, and additionally holds the Neumann condition in (2.27d) for those faces that belong to the Neumann boundary. It remains to define the Dirichlet boundary condition, which is weakly enforced by

$$\int_{\partial\Omega_D} \mu(\lambda - g_D) \, \mathrm{d}s = 0, \qquad \forall \mu \in L^2(\partial\Omega_D). \tag{2.34}$$

The weak form of the discontinuous Galerkin problem reads:

Find $(u, \boldsymbol{q}, \lambda) \in \left(L^2(\Omega), \left[L^2(\Omega)\right]^{n_{sd}}, L^2(\Gamma)\right)$ *such that:*

$$\sum_{\Omega^e \in \Omega} \int_{\partial\Omega^e} v \widetilde{\boldsymbol{q}} \cdot \boldsymbol{n} \, \mathrm{d}s - \sum_{\Omega^e \in \Omega} \int_{\Omega^e} (\nabla v) \cdot \boldsymbol{q} \, \mathrm{d}\boldsymbol{x} = \sum_{\Omega^e \in \Omega} \int_{\Omega^e} v f \, \mathrm{d}\boldsymbol{x}, \quad \forall v \in L^2(\Omega);$$

$$\sum_{\Omega^e \in \Omega} \int_{\Omega^e} \boldsymbol{w} \cdot \mathbf{D}^{-1} \cdot \boldsymbol{q} \, \mathrm{d}\boldsymbol{x} + \sum_{\Omega^e \in \Omega} \int_{\partial\Omega^e} (\boldsymbol{w} \cdot \boldsymbol{n}) \widetilde{u} \, \mathrm{d}s - \sum_{\Omega^e \in \Omega} \int_{\Omega^e} (\nabla \cdot \boldsymbol{w}) u \, \mathrm{d}\boldsymbol{x} = 0, \qquad \forall \boldsymbol{w} \in [L^2(\Omega)]^{n_{sd}};$$

$$\int_{\partial\Omega_D} \mu(\lambda - g_D) \, \mathrm{d}s = 0, \qquad \forall \mu \in L^2(\partial\Omega_D);$$

$$\sum_{\Omega^e \in \Omega} \int_{\partial\Omega^e} \mu \widetilde{\boldsymbol{q}} \cdot \boldsymbol{n} \, \mathrm{d}s = \int_{\partial\Omega_N} \mu g_N \, \mathrm{d}s, \qquad \forall \mu \in L^2(\Gamma \backslash \partial\Omega_D);$$

according to the definition of $\widetilde{\boldsymbol{q}}$*:*

$$\widetilde{\boldsymbol{q}}^e(\boldsymbol{x}) := \boldsymbol{q}^e(\boldsymbol{x}) + \tau \cdot (u^e(\boldsymbol{x}) - \lambda^e(\boldsymbol{x})) \cdot \boldsymbol{n}^e \qquad \text{on } \partial\Omega^e. \tag{2.35}$$

2.3.2 Discretization

The domain Ω is discretized in several smaller elements Ω^e. Since the solution is discontinuous between elements, the scalar unknown $u(\boldsymbol{x})$ is locally discretized in $u^e(\boldsymbol{x})$ for all the elements of the mesh:

$$u^e(\boldsymbol{x}) \approx \sum_{j=1}^{N_{en}} \phi_j^e(\boldsymbol{x}) \hat{u}_j^e, \tag{2.36}$$

being $\hat{u}_j^e$ the value of the scalar unknown u at the node $\boldsymbol{x}_j^e$ of the e-th element, $\hat{u}_j^e = u(\boldsymbol{x}_j^e)$. Here, N_{en} refers to the number of elemental nodes. The main difference with the CG formulation is that the Dirichlet condition does *not* prescribe u values, but the values of its numerical trace λ. In this way, *all* the element nodes will hold an *unknown* value for u.

The discretization of the flux $\boldsymbol{q}^e(\boldsymbol{x})$ is defined in terms of the vectors $\boldsymbol{e}_k$, $k = 1 \dots n_{sd}$, which are the k columns of the identity matrix $I_{n_{sd}}$. In this definition, n_{sd} is the number of spatial dimensions of the problem (2 for the 2D case and 3 for 3D case). The flux is discretized in the following way:

$$\boldsymbol{q}^e(\boldsymbol{x}) \approx \sum_{k=1}^{n_{sd}} \sum_{j=1}^{N_{en}} \phi_j^e(\boldsymbol{x}) \hat{q}_{kj}^e \boldsymbol{e}_k, \tag{2.37}$$

being $\hat{q}_{kj}^e$ the value of the component k of the flux $\boldsymbol{q}$ at the node $\boldsymbol{x}_j^e$ of the e-th element, $\hat{q}_{kj}^e = \boldsymbol{q}(\boldsymbol{x}_j^e)[k]$. Since we implement the Galerkin method, we approximate the spaces of test functions $L^2(\Omega^e)$ and $[L^2(\Omega^e)]^{n_{sd}}$ as:

$$V = \text{span } \{\phi_i^e(\boldsymbol{x}), \quad \forall i = 1 \dots N_{en}\}, \tag{2.38a}$$

$$W = \text{span } \{\phi_i^e(\boldsymbol{x}) \boldsymbol{e}_m, \quad \forall i = 1 \dots N_{en}, \quad \forall m = 1 \dots n_{sd}\}, \tag{2.38b}$$

respectively. Thus, we set the test functions $v(\boldsymbol{x})$ and $\boldsymbol{w}(\boldsymbol{x})$ as

$$v^e(\boldsymbol{x}) = \phi_i^e(\boldsymbol{x}), \qquad \forall i = 1 \dots N_{en}; \tag{2.39a}$$

$$\boldsymbol{w}^e(\boldsymbol{x}) = \phi_i^e(\boldsymbol{x}) \boldsymbol{e}_m, \qquad \forall i = 1 \dots N_{en}, \quad \forall m = 1 \dots n_{sd}. \tag{2.39b}$$

Therefore, the gradient of $v(\boldsymbol{x})$ is:

$$\nabla v^e(\boldsymbol{x}) = \nabla \phi_i^e(\boldsymbol{x}), \qquad \forall i = 1 \dots N_{en}; \tag{2.40}$$

and the divergence of $\boldsymbol{w}(\boldsymbol{x})$:

$$\nabla \cdot \boldsymbol{w}^e(\boldsymbol{x}) = \nabla \cdot (\phi_i^e(\boldsymbol{x}) \boldsymbol{e}_m) = \frac{\partial}{\partial x_m} \phi_i^e(\boldsymbol{x}), \qquad \forall i = 1 \dots N_{en}, \quad \forall m = 1 \dots n_{sd}. \tag{2.41}$$

We have to discretize as well the functions defined along the *boundary* of each element, $\partial\Omega^e$, which are the numerical trace $\lambda^e(\boldsymbol{x})$ and its corresponding test function $\mu^e(\boldsymbol{x})$. In order to do so, we will make use of a new kind of shape functions $\psi_j^e(\boldsymbol{x})$ defined along the boundary $\partial\Omega^e$. These $\psi_j^e(\boldsymbol{x})$ shape functions have the same polinomial degree p as $\phi_j^e(\boldsymbol{x})$.

Thus, the discretization of $\lambda^e(\boldsymbol{x})$ reads

$$\lambda^e(\boldsymbol{x}) \approx \sum_{s=1}^{N_{ef}} \sum_{j=1}^{N_{fn}} \psi_{sj}^e(\boldsymbol{x}) \hat{\lambda}_{sj}^e; \tag{2.42}$$

where N_{ef} is to the number of element faces, N_{fn} is the number of face nodes, and $\hat{\lambda}_{sj}^e$ refers to the value of the numerical trace λ at the node $\boldsymbol{x}_j^{s,e}$ of the s-th face in the e-th element. The test function $\mu^e(\boldsymbol{x})$ is discretized in a similiar way for each element face:

$$\mu^e(\boldsymbol{x}) \approx \sum_{l=1}^{N_{ef}} \psi_{li}^e(\boldsymbol{x}), \qquad \forall i = 1 \dots N_{fn}. \tag{2.43}$$

We are ready to come back to the weak form. Inserting the definition of the numerical trace (2.31) and the discretizations (2.36) to (2.43) in the particularization for the e-th element of the weak form in (2.30), we end up with

$$\int_{\partial\Omega^e} \phi_i^e \left(\phi_j^e \hat{q}_{kj}^e \boldsymbol{e}_k + \tau \cdot (\phi_j^e \hat{u}_j^e - \psi_{sj}^e \hat{\lambda}_{sj}^e) \cdot \boldsymbol{n}^e \right) \cdot \boldsymbol{n}^e \, \mathrm{d}s - \int_{\Omega^e} (\nabla \phi_i^e) \cdot \phi_j^e \hat{q}_{kj}^e \boldsymbol{e}_k \, \mathrm{d}\boldsymbol{x} = \int_{\Omega^e} \phi_i^e f \, \mathrm{d}\boldsymbol{x}$$
$$\forall e = 1 \dots N_{el}, \quad \forall i = 1 \dots N_{en}; \tag{2.44a}$$

$$\int_{\Omega^e} \phi_i^e \boldsymbol{e}_m \cdot \mathbf{D}^{-1} \phi_j^e \hat{q}_{kj}^e \boldsymbol{e}_k \, \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega^e} (\phi_i^e \boldsymbol{e}_m \cdot \boldsymbol{n}^e) \psi_{sj}^e \hat{\lambda}_{sj}^e \, \mathrm{d}s - \int_{\Omega^e} \frac{\partial \phi_i^e}{\partial x_m} \phi_j^e \hat{u}_j^e \, \mathrm{d}\boldsymbol{x} = 0$$
$$\forall e = 1 \dots N_{el}, \quad \forall i = 1 \dots N_{en}, \quad \forall m = 1 \dots n_{sd}; \tag{2.44b}$$

$$\int_{\partial\Omega^e} \psi_{li}^e \psi_{sj}^e \hat{\lambda}_{sj}^e \, \mathrm{d}s = \int_{\partial\Omega^e} \psi_{li}^e g_D \, \mathrm{d}s, \qquad \forall i = 1 \dots N_{fn}, \quad \forall l = 1 \dots N_{ef_D}; \tag{2.44c}$$

$$\int_{\partial\Omega^e} \psi_{li}^e \left(\phi_j^e \hat{q}_{kj}^e \boldsymbol{e}_k + \tau \cdot (\phi_j^e \hat{u}_j^e - \psi_{sj}^e \hat{\lambda}_{sj}^e) \cdot \boldsymbol{n}^e \right) \cdot \boldsymbol{n}^e \, \mathrm{d}s = \int_{\partial\Omega^e} \psi_{li}^e g_N \, \mathrm{d}s,$$
$$\forall i = 1 \dots N_{fn}, \quad \forall l = (N_{ef_D} + 1) \dots N_{ef}; \tag{2.44d}$$

from here to the end of this document, repeated indices account for the Einstein convention. Here, N_{ef_D} refers to the number of elemental faces that belong to the Dirichlet boundary $\partial\Omega_D$; in this way, the ψ_{li}^e shape functions in equation (2.44c) refer only to the Dirichlet faces, whereas ψ_{li}^e in (2.44d) refer to all the restant faces.

In order to reorganize terms and simplify the formulation in (2.44), it is interesting to

note that (i) $\boldsymbol{n}^e \cdot \boldsymbol{n}^e = \|\boldsymbol{n}^e\|_{L_2(\Omega)}{}^2 = 1$, (ii) $\boldsymbol{e}_k \cdot \boldsymbol{a} = \boldsymbol{a} \cdot \boldsymbol{e}_k = a_k$ for any vector $\boldsymbol{a}$ of the same length of $\boldsymbol{e}_k$, (iii) $\boldsymbol{e}_m \cdot \mathbf{A} \cdot \boldsymbol{e}_k = \mathrm{A}_{mk}$ for any matrix $\mathbf{A}$ with the same number of rows as the length of $\boldsymbol{e}_m$ and the same number of columns as the length of $\boldsymbol{e}_k$, and (iv) the terms τ, $\hat{u}^e_j$, $\hat{q}^e_{kj}$ and $\hat{\lambda}^e_{sj}$ do not depend on the coordinates $\boldsymbol{x}$, and therefore they can get out of the integrals.

After some straightforward simplifications on (2.44), we get the final expression of the weak form discretization of the LDG-H scheme for the elliptic problem:

Find $u_1 \dots u_{N_{en}}$, $q_{1,1} \dots q_{n_{sd},N_{en}}$ *and* $\lambda_{1,1} \dots \lambda_{N_{ef},N_{fn}}$, *such that:*

$$\tau \int_{\Omega^e} \phi^e_i \phi^e_j \,\mathrm{d}s \cdot \hat{u}^e_j + \int_{\partial\Omega^e} \phi^e_i \phi^e_j n^e_k \,\mathrm{d}s \cdot \hat{q}^e_{kj} - \int_{\Omega^e} \frac{\partial \phi^e_i}{\partial x_k} \cdot \phi^e_j \,\mathrm{d}\boldsymbol{x} \cdot \hat{q}^e_{kj} - \tau \int_{\partial\Omega^e} \phi^e_i \cdot \psi^e_{sj} \,\mathrm{d}s \cdot \hat{\lambda}^e_{sj} = \int_{\Omega^e} \phi^e_i f \,\mathrm{d}\boldsymbol{x}, \qquad \forall e = 1 \dots N_{el}, \quad \forall i = 1 \dots N_{en}; \tag{2.45a}$$

$$- \int_{\Omega^e} \frac{\partial \phi^e_i}{\partial x_m} \phi^e_j \,\mathrm{d}\boldsymbol{x} \cdot \hat{u}^e_j + \int_{\Omega^e} \phi^e_i \left[\mathbf{D}^{-1}\right]_{mk} \phi^e_j \,\mathrm{d}\boldsymbol{x} \cdot \hat{q}^e_{kj} + \int_{\partial\Omega^e} \phi^e_i \psi^e_{sj} n^e_m \,\mathrm{d}s \cdot \hat{\lambda}^e_{sj} = 0, \qquad \forall e = 1 \dots N_{el}, \quad \forall i = 1 \dots N_{en}, \quad \forall m = 1 \dots n_{sd}; \tag{2.45b}$$

$$\int_{\partial\Omega^e} \psi^e_{li} \psi^e_{sj} \,\mathrm{d}s \cdot \hat{\lambda}^e_{sj} = \int_{\partial\Omega^e} \psi^e_{li} g_D \,\mathrm{d}s, \qquad \forall i = 1 \dots N_{fn}, \quad \forall l = 1 \dots N_{ef_D}; \tag{2.45c}$$

$$\tau \int_{\partial\Omega^e} \psi^e_{li} \phi^e_j \,\mathrm{d}s \cdot \hat{u}^e_j + \int_{\partial\Omega^e} \psi^e_{li} \phi^e_j n^e_k \,\mathrm{d}s \cdot \hat{q}^e_{kj} - \tau \int_{\partial\Omega^e} \psi^e_{li} \cdot \psi^e_{sj} \,\mathrm{d}s \cdot \hat{\lambda}^e_{sj} = \int_{\partial\Omega^e} \psi^e_{li} g_N \,\mathrm{d}s \qquad \forall i = 1 \dots N_{fn}, \quad \forall l = (N_{ef_D} + 1) \dots N_{ef}. \tag{2.45d}$$

Equations (2.45a) and (2.45b) are known as the *local solvers*, since they solve the unknowns $\hat{\boldsymbol{u}}^e$ ($\hat{\boldsymbol{u}}^e[j] = \hat{u}^e_j$) and $\hat{\boldsymbol{q}}_{\boldsymbol{k}}{}^e$ ($\hat{\boldsymbol{q}}_{\boldsymbol{k}}{}^e[j] = \hat{q}^e_{kj}$) within the element e in terms of the information of the traces $\hat{\boldsymbol{\lambda}}_{\boldsymbol{s}}{}^e$ ($\hat{\boldsymbol{\lambda}}_{\boldsymbol{s}}{}^e[j] = \hat{\lambda}^e_{sj}$) along its boundaries s. Equation (2.45c) simply forces the numerical trace of the scalar unknown $\hat{\boldsymbol{\lambda}}_{\boldsymbol{s}}{}^e$ to be the projection of the Dirichlet condition along $\partial\Omega^e$ for those specific traces that belong to the Dirichlet boundary. Finally, equation (2.45d) is known as the *transmission condition*, because it determines the relationship between the traces $\hat{\boldsymbol{\lambda}}_{\boldsymbol{s}}{}^e$ surrounding the e-th element, and therefore *couples* the information of all the numerical traces of the mesh once the elemental contributions of all the elements have been assembled.

The weak form in (2.45) can be expressed in matrix form according to the next definitions:

$$\begin{aligned}
\mathbb{M}^e_{mk}[i,j] &:= \int_{\Omega^e} \phi^e_i \left[\mathbf{D}^{-1}\right]_{mk} \phi^e_j \,\mathrm{d}\boldsymbol{x}, & \mathbb{P}^e_{sk}[i,j] &:= \int_{\Gamma^e_s} \phi^e_i \phi^e_j n^{e,s}_k \,\mathrm{d}s,\\
\mathbb{D}^e_k[i,j] &:= \int_{\Omega^e} \frac{\partial \phi^e_i}{\partial x_k} \phi^e_j \,\mathrm{d}\boldsymbol{x}, & \mathbb{Q}^e_{ks}[i,j] &:= \int_{\Gamma^e_s} \phi^e_i \psi^e_{sj} n^{e,s}_k \,\mathrm{d}s,\\
\mathbb{E}^e_s[i,j] &:= \int_{\Gamma^e_s} \phi^e_i \phi^e_j \,\mathrm{d}s, & \mathbb{f}^e[i] &:= \int_{\Omega^e} \phi^e_i f \,\mathrm{d}\boldsymbol{x},\\
\mathbb{F}^e_s[i,j] &:= \int_{\Gamma^e_s} \phi^e_i \psi^e_{sj} \,\mathrm{d}s, & \mathbb{d}^e_s[i] &:= \int_{\Gamma^e_s} \psi^e_{si} g_D \,\mathrm{d}s,\\
\mathbb{G}^e_s[i,j] &:= \int_{\Gamma^e_s} \psi^e_{si} \psi^e_{sj} \,\mathrm{d}s, & \mathbb{g}^e_s[i] &:= \int_{\Gamma^e_s} \psi^e_{si} g_N \,\mathrm{d}s.
\end{aligned} \tag{2.46}$$

Then, the matrix form of the contribution of the e-th element is

$$\tau \left(\sum_{s=1}^{N_{ef}} \mathbb{E}^e_s \right) \cdot \hat{\boldsymbol{u}}^e + \left(\sum_{s=1}^{N_{ef}} \mathbb{P}^e_{sk} - \mathbb{D}^e_k \right) \cdot \hat{\boldsymbol{q}_k}^e - \tau \mathbb{F}^e_s \cdot \hat{\boldsymbol{\lambda}_s}^e = \mathbb{f}^e; \tag{2.47a}$$

$$-\mathbb{D}^e_m \cdot \hat{\boldsymbol{u}}^e + \mathbb{M}^e_{mk} \cdot \hat{\boldsymbol{q}_k}^e + \mathbb{Q}^e_{ms} \cdot \hat{\boldsymbol{\lambda}_s}^e = \boldsymbol{0}, \quad \forall m = 1 \dots n_{sd}; \tag{2.47b}$$

$$\mathbb{G}^e_s \cdot \hat{\boldsymbol{\lambda}_s}^e = \mathbb{d}^e_s, \quad \forall s = 1 \dots N_{ef_D}; \tag{2.47c}$$

$$\tau [\mathbb{F}^e_s]^T \cdot \hat{\boldsymbol{u}}^e + [\mathbb{Q}^e_{ks}]^T \cdot \hat{\boldsymbol{q}_k}^e - \tau \mathbb{G}^e_s \cdot \hat{\boldsymbol{\lambda}_s}^e = \mathbb{g}^e_s, \quad \forall s = (N_{ef_D} + 1) \dots N_{ef}. \tag{2.47d}$$

To get to (2.47), two properties have been considered:

(i) The integrals along the boundary of the element have been decomposed in the sum of the integrals along all the elemental faces

$$\int_{\partial\Omega^e} f \,\mathrm{d}s = \sum_{s=1}^{N_{ef}} \int_{\Gamma^e_s} f \,\mathrm{d}s, \qquad \text{for any given } f \in L^2(\partial\Omega^e);$$

(ii) The scalar product in Γ_s between the test function of the l-th numerical trace and any other function f vanishes if $l \neq s$,

$$\int_{\partial\Gamma^e_s} \psi^e_{li} f \,\mathrm{d}s = 0 \qquad \text{if } l \neq s.$$

The combination of (i) and (ii) leads to

$$\int_{\partial\Omega^e} \psi^e_{li} f \,\mathrm{d}s = \int_{\partial\Gamma^e_l} \psi^e_{li} f \,\mathrm{d}s,$$

which can be particularized in

$$\int_{\partial\Omega^e} \psi_{li}^e \psi_{sj}^e \,\mathrm{d}s = \int_{\partial\Gamma_s^e} \psi_{li}^e \psi_{sj}^e \delta_{ls} \,\mathrm{d}s.$$

The formulation (2.47) can get even more simplified grouping the matrices in (2.46) in larger matrices $\mathcal{A}^e$, $\mathcal{B}^e$, $\mathcal{C}^e$ and $\mathcal{D}^e$ in the following way:

$$\mathcal{A}^e := \begin{pmatrix} \tau \sum_{s=1}^{N_{ef}} \mathbb{E}_s^e & \sum_{s=1}^{N_{ef}} \mathbb{P}_{s1}^e - \mathbb{D}_1^e & \cdots & \sum_{s=1}^{N_{ef}} \mathbb{P}_{sn_{sd}}^e - \mathbb{D}_{n_{sd}}^e \\ -\mathbb{D}_1^e & \mathbb{M}_{11}^e & \cdots & \mathbb{M}_{1n_{sd}}^e \\ \vdots & \vdots & \ddots & \vdots \\ -\mathbb{D}_{n_{sd}}^e & \mathbb{M}_{n_{sd}1}^e & \cdots & \mathbb{M}_{n_{sd}n_{sd}}^e \end{pmatrix}; \tag{2.48a}$$

$$\mathcal{B}^e := \begin{pmatrix} -\tau\mathbb{F}_1^e & \cdots & -\tau\mathbb{F}_{N_{ef}}^e \\ \mathbb{Q}_{11}^e & \cdots & \mathbb{Q}_{1N_{ef}}^e \\ \vdots & \ddots & \vdots \\ \mathbb{Q}_{n_{sd}1}^e & \cdots & \mathbb{Q}_{n_{sd}N_{ef}}^e \end{pmatrix}; \tag{2.48b}$$

$$\mathcal{C}^e := \begin{pmatrix} \mathcal{C}_1^e \\ \vdots \\ \mathcal{C}_{N_{ef}}^e \end{pmatrix}, \text{ where } \mathcal{C}_s^e := \begin{cases} \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} & \text{if } \Gamma_s^e \in \partial\Omega_D, \\ \begin{pmatrix} \tau[\mathbb{F}_s^e]^T & [\mathbb{Q}_{1s}^e]^T & \cdots & [\mathbb{Q}_{n_{sd}s}^e]^T \end{pmatrix} & \text{if } \Gamma_s^e \notin \partial\Omega_D; \end{cases} \tag{2.48c}$$

$$\mathcal{D}^e := \begin{pmatrix} \mathcal{D}_1^e & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{D}_{N_{ef}}^e \end{pmatrix}, \text{ where } \mathcal{D}_s^e := \begin{cases} \mathbb{G}_s^e & \text{if } \Gamma_s^e \in \partial\Omega_D, \\ -\tau\mathbb{G}_s^e & \text{if } \Gamma_s^e \notin \partial\Omega_D. \end{cases} \tag{2.48d}$$

With the definitions in (2.48), it is straightforward to write the matrix form of the local solver

$$\mathcal{A}^e \cdot \begin{bmatrix} \hat{\boldsymbol{u}}^e \\ \hat{\boldsymbol{q}}_1{}^e \\ \vdots \\ \hat{\boldsymbol{q}}_{\boldsymbol{n_{sd}}}{}^e \end{bmatrix} + \mathcal{B}^e \cdot \begin{bmatrix} \hat{\boldsymbol{\lambda}}_1{}^e \\ \vdots \\ \hat{\boldsymbol{\lambda}}_{\boldsymbol{N_{ef}}}{}^e \end{bmatrix} = \begin{bmatrix} \mathbb{f}^e \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \tag{2.49}$$

as well as the matrix form of the transmission condition

$$\mathcal{C}^e \cdot \begin{bmatrix} \hat{\boldsymbol{u}}^e \\ \hat{\boldsymbol{q}}_1{}^e \\ \vdots \\ \hat{\boldsymbol{q}}_{\boldsymbol{n_{sd}}}{}^e \end{bmatrix} + \mathcal{D}^e \cdot \begin{bmatrix} \hat{\boldsymbol{\lambda}}_1{}^e \\ \vdots \\ \hat{\boldsymbol{\lambda}}_{\boldsymbol{N_{ef}}}{}^e \end{bmatrix} = \begin{bmatrix} \mathbb{t}_1^e \\ \vdots \\ \mathbb{t}_{N_{ef}}^e \end{bmatrix}, \text{ where } \mathbb{t}_s^e := \begin{cases} \mathbb{d}_s^e & \text{if } \Gamma_s^e \in \partial\Omega_D^e, \\ \mathbb{g}_s^e & \text{if } \Gamma_s^e \notin \partial\Omega_D^e. \end{cases} \tag{2.50}$$

In a similar way, we can write the e-th elemental contributions as

$$\mathbf{K}^e := \begin{pmatrix} \mathcal{A}^e & \mathcal{B}^e \\ \mathcal{C}^e & \mathcal{D}^e \end{pmatrix}, \qquad \boldsymbol{a}^e := \begin{bmatrix} \hat{\boldsymbol{u}}^e \\ \hat{\boldsymbol{q}}_1{}^e \\ \vdots \\ \hat{\boldsymbol{q}}_{\boldsymbol{n_{sd}}}{}^e \\ \hat{\boldsymbol{\lambda}}_1{}^e \\ \vdots \\ \hat{\boldsymbol{\lambda}}_{\boldsymbol{N_{ef}}}{}^e \end{bmatrix}, \qquad \boldsymbol{f}^e := \begin{bmatrix} \mathbb{f}^e \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbb{t}_1^e \\ \vdots \\ \mathbb{t}_{N_{ef}}^e \end{bmatrix}. \tag{2.51}$$

Finally, the assembly process of $\mathbf{K}$ and $\boldsymbol{f}$ for the whole system is performed as usual

$$\mathbf{K} = \bigwedge_{e,\Omega^e \in \Omega} \mathbf{K}^e, \qquad \boldsymbol{f} = \bigwedge_{e,\Omega^e \in \Omega} \boldsymbol{f}^e, \tag{2.52}$$

so the final form of the linear system for the whole problem is

$$\mathbf{K} \cdot \boldsymbol{a} = \boldsymbol{f}. \tag{2.53}$$

The assembly process has to be carried out cautiously, specially when assembling the numerical traces $\hat{\boldsymbol{\lambda}}_{\boldsymbol{s}}{}^e$, since the local numbering order of the nodes $\boldsymbol{x}_j^{s,e}$ along it may be reversed from one element to the adjacent one (the local numbering of the nodes of the numerical trace *depends* on the element it is considered to belong to).

Note that vector $\boldsymbol{a}$ can be found out solving the global system, and then the scalar unknown, the flux and (eventually) the numerical trace can be retrieved using equations (2.36), (2.37) and (2.42) for each element, respectively.

2.3.3 Post-process

A special feature of the mixed methods is that the approximate solution can be *locally post-processed* in order to obtain a new approximation of the scalar unknown converging with an additional order $\mathcal{O}(h^{p+2})$ [6, 8, 26]. This so called *superconvergence* is achieved for $p > 0$ if the flux converges with order $(p+1)$ and the average of the scalar unknown on each element superconverges with order $(p+2)$; see details in [15].

Let us denote $u^e_{\text{post}}(\boldsymbol{x})$ the post-processed $u^e(\boldsymbol{x})$. The idea is to take $\boldsymbol{q}^e(\boldsymbol{x})$ and find an $u^e_{\text{post}}(\boldsymbol{x})$ such that $\boldsymbol{q}^e(\boldsymbol{x}) = -\mathbf{D}(\boldsymbol{x})\nabla u^e_{\text{post}}(\boldsymbol{x})$. Since $\boldsymbol{q}(\boldsymbol{x})$ is interpolated with polynomial shape functions of degree p, $u^e_{\text{post}}(\boldsymbol{x})$ has to be interpolated with polynomials of degree

$(p+1)$. This problem is weakly defined by imposing:

$$\int_{\Omega^e} \nabla w^e \cdot \nabla u^e_{\text{post}} \, \mathrm{d}\boldsymbol{x} = -\int_{\Omega^e} \nabla w^e \cdot \mathbf{D}^{-1} \boldsymbol{q}^e \, \mathrm{d}\boldsymbol{x}, \qquad \forall w \in L^2(\Omega). \tag{2.54}$$

The discretization of w^e and u^e_{post} with the usual shape functions of degree $(p+1)$ leads to an underdetermined system of linear equations, which requires an additional equation to be added. This equation is

$$\int_{\Omega^e} \left(u^e_{\text{post}} - u^e\right) \, \mathrm{d}\boldsymbol{x} = 0, \tag{2.55}$$

which forces the average of $u^e_{\text{post}}(\boldsymbol{x})$ and the average of $u^e(\boldsymbol{x})$ to coincide within the e-th element.

The new post-processed $u_{\text{post}}(\boldsymbol{x})$ is obtained repeating this procedure *locally* for each element of the mesh.

2.4 HDG Formulation

This section shows the hybridization of the LDG-H formulation seen in section 2.3, using the same Schur Complement technique exposed in 2.2. The LDG-H formulation has been designed in order to be hybridizable, and here we take advantage of this property to build up a faster and less memory-demanding solver, called *HDG*. Similarly as in 2.2.2, it is intended to hybridize the local contributions of each element, and not the whole assembled system.

We define the set of primary unknowns $\boldsymbol{a}^e_\lambda$ which contains the unknown λ, and the set of secondary unknowns $\boldsymbol{a}^e_h$ which contains u and $\boldsymbol{q}$:

$$\boldsymbol{a}^e_\lambda := \begin{bmatrix} \hat{\boldsymbol{\lambda}}_1{}^e \\ \vdots \\ \hat{\boldsymbol{\lambda}}_{\boldsymbol{N_{ef}}}{}^e \end{bmatrix}, \qquad \boldsymbol{a}^e_h := \begin{bmatrix} \hat{\boldsymbol{u}}^e \\ \hat{\boldsymbol{q}}_1{}^e \\ \vdots \\ \hat{\boldsymbol{q}}_{\boldsymbol{n_{sd}}}{}^e \end{bmatrix}. \tag{2.56}$$

This definition divides the matrix $\mathbf{K}^e$ and the vector $\boldsymbol{f}^e$ in four and two blocks, respectively:

$$\mathbf{K}^e := \begin{pmatrix} \mathbf{K}^e_{hh} & \mathbf{K}^e_{h\lambda} \\ \mathbf{K}^e_{\lambda h} & \mathbf{K}^e_{\lambda\lambda} \end{pmatrix}, \qquad \boldsymbol{a}^e := \begin{bmatrix} \boldsymbol{a}^e_h \\ \boldsymbol{a}^e_\lambda \end{bmatrix}, \qquad \boldsymbol{f}^e := \begin{bmatrix} \boldsymbol{f}^e_h \\ \boldsymbol{f}^e_\lambda \end{bmatrix}. \tag{2.57}$$

Note that both u and q are not directly related between elements, thus leading to a block-diagonal $\mathbf{K}^e_{hh}$ submatrix. From (2.49) and (2.50), we find the next identities:

$$\begin{aligned} \mathbf{K}^e_{hh} &= \mathcal{A}^e, \\ \mathbf{K}^e_{h\lambda} &= \mathcal{B}^e, \\ \mathbf{K}^e_{\lambda h} &= \mathcal{C}^e, \\ \mathbf{K}^e_{\lambda\lambda} &= \mathcal{D}^e, \end{aligned} \qquad \boldsymbol{f}^e_h = \begin{bmatrix} \mathbb{f}^e \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \qquad \boldsymbol{f}^e_\lambda = \begin{bmatrix} \mathbb{t}^e_1 \\ \vdots \\ \mathbb{t}^e_{N_{ef}} \end{bmatrix}. \tag{2.58}$$

Therefore, we can say that the equations in (2.57) that will remain after the hybridization are the ones that belong to the *transmission condition* (2.50), and the ones that will be hybridized are those belonging to the *local solver* (2.49), which is exactly what was meant to do. The hybridization of the elemental contributions in this way is possible because the matrix $\mathcal{A}^e$ is always non-singular. This is justified by [7, 15], since every local solver involves the DG discretization of an elemental domain with Dirichlet boundary conditions, and therefore each local elemental problem is well-posed and invertible.

We can apply then the static condensation procedure shown in (2.23) to this formulation in order to get the hybridized version of the transmission condition:

$$\{\mathcal{D}^e - \mathcal{C}^e \cdot (\mathcal{A}^e)^{-1} \cdot \mathcal{B}^e\} \cdot \boldsymbol{a}^e_\lambda = \boldsymbol{f}^e_\lambda - \mathcal{C}^e \cdot (\mathcal{A}^e)^{-1} \cdot \boldsymbol{f}^e_h. \tag{2.59}$$

The assembly process of $\mathbf{K}$ and $\boldsymbol{f}$ (let us label them as $\mathbf{K}_\lambda$ and $\boldsymbol{f}_\lambda$ in this hybridized formulation) is performed in the following way:

$$\mathbf{K}_\lambda = \bigwedge_{\Omega^e \in \Omega} \mathcal{D}^e - \mathcal{C}^e \cdot (\mathcal{A}^e)^{-1} \cdot \mathcal{B}^e, \qquad \boldsymbol{f}_\lambda = \bigwedge_{\Omega^e \in \Omega} \boldsymbol{f}^e_\lambda - \mathcal{C}^e \cdot (\mathcal{A}^e)^{-1} \cdot \boldsymbol{f}^e_h, \tag{2.60}$$

so the final form of the linear system for the condensed problem is

$$\mathbf{K}_\lambda \cdot \boldsymbol{a}_\lambda = \boldsymbol{f}_\lambda. \tag{2.61}$$

The unknowns defined within the elements (the scalar unknown and the flux) have been *hybridized* and are not present in (2.61). The vector $\boldsymbol{a}_\lambda$ is obtained once the linear system in (2.61) is solved. The value of the scalar unknown and the flux within the elements can be computed *locally* retrieving the expression of the local solver in terms of the numerical trace:

$$\boldsymbol{a}^e_h = (\mathcal{A}^e)^{-1} \cdot \boldsymbol{f}^e_h - (\mathcal{A}^e)^{-1} \cdot \mathcal{B}^e \cdot \boldsymbol{a}^e_\lambda. \tag{2.62}$$

Once the scalar unknown, the flux and the traces are known, it is possible to perform the *local post-processing* of the solution in the same way as it is done in section 3.2.4.

To illustrate the reduction of the size of the linear system due to its hybridization, we consider a structured quadrilateral mesh, such that $N_{el} = N_d \cdot N_d$, being N_{el} the number of elements of the mesh and N_d the number of divisions of both $x-$ and $y-$boundaries. The linear system in (2.61) has $Neq_\lambda = N_\lambda(p+1)$ equations, being N_λ the total number of faces of the mesh. Prior to the hybridization process, the system had $Neq_h = 3N_{el}(p+1)^2$ *further* equations. Taking into account that for this kind of quadrilateral meshes the number of faces is related with the number of elements as:

$$N_\lambda = 2\left(N_{el} + N_d\right),$$

it can be shown that, for fine meshes, the reduction of the number of equations in the global system due to the hybridization process can reach values up to

$$\lim_{N_{el}\to\infty}\left(\frac{Neq_h}{Neq_\lambda + Neq_h}\right) = \ldots = \frac{3p^2 - 6p + 3}{3p^2 + 8p + 5}.$$

This means a reduction of the number of equations of 75% for $p = 1$, 86% for $p = 3$ or 90% for $p = 5$, to name a few examples. It is clear, then, that the hybridization technique reduces *drastically* the size of the linear system for high interpolation degrees. Note that for 3D problems, the reduction is even higher, since the ratio between the number of inner nodes of a given element and the number of their face nodes increases with p faster than in the 2D case.

Chapter 3

Implementation and Computational Strategy

The objective of this chapter is to present the most relevant aspects of the developed implementation of the formulations presented in chapter 2.

This chapter is structured as follows. Section 3.1 explains the different softwares that have been used throughout this thesis. Section 3.2 deals with the implementation of the finite element code. This section is the most relevant one in terms of computational engineering, and focuses on specific parts of the code that are worth commenting, such as the building of the linear systems (section 3.2.2), the solvers (section 3.2.3) and the post-process (section 3.2.4). Finally, the required operations after the problem has been executed (error computation and files export) are explained in section 3.3.

3.1 Software Overview

Several softwares have been used in this thesis. Figure 3.1 shows the general flowchart of the thesis implementation and the relationship between the different softwares in it. The implementation consists of three main sections:

The generation of the meshes is *not* inside the scope of this thesis. In order to generate the meshes, an in-house software called '`ez4u`' [17] has been used. This software is able to generate 2D/3D high-order meshes with different geometric elements (triangles, quadrilaterals and hexaedra are used here) according to the methodologies described in [11, 12, 25].

The basic information of a mesh, which is:

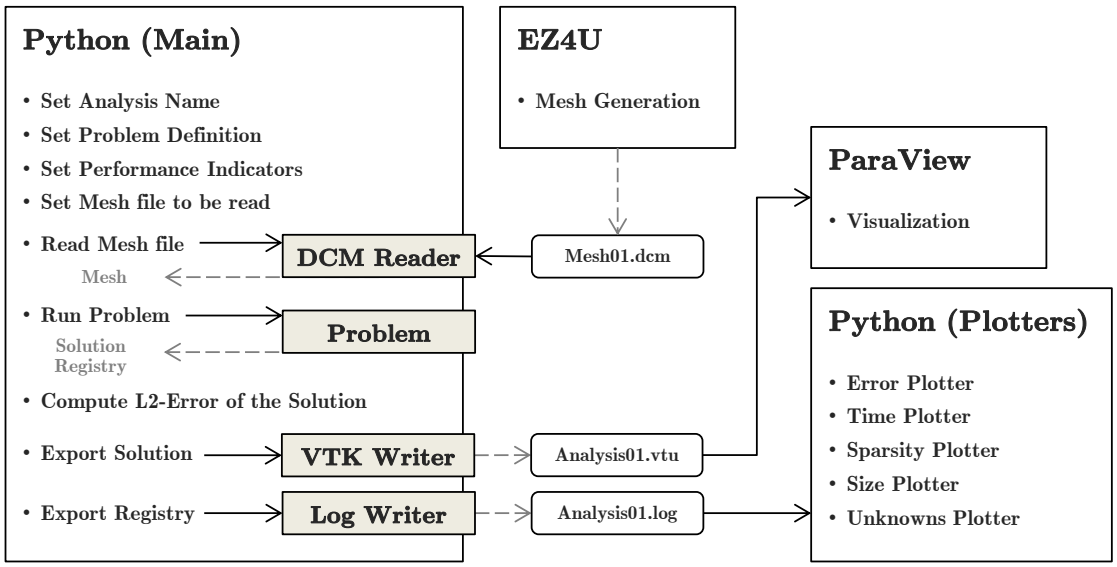


FIGURE 3.1: Implementation flowchart.

- The coordinates matrix
- The connectivity matrix
- The location of the boundary conditions (e.g. Dirichlet and Neumann ones)

is stored in a file according to the `.dcm` mesh format. Therefore, our code implementation (see section 3.1-Finite element code) needs a *DCM Reader* which understands the information encoded in the `.dcm` files correctly. This mesh reader was already implemented by my mentors when I first received the code, so *it's not part of my own work.*

The output of this process is a `Python` [22] instance of a `Mesh` class. It contains the coordinates matrix, the connectivity matrix, the prescribed conditions and the master element, which is fundamental in order to get the elemental contributions: it contains information about the geometry of the reference element, the shape functions and its derivatives, the weights and points of the numerical quadrature required to integrate any function within its domain and the way to compute the normal vectors at a given point.

The finite element code able to solve elliptic diffusion problems has been implemented in '`Python`'. This code includes the mesh reading, the building of the linear system (the computation of the elemental contributions and the assembly process), the solvers and the export operations (see sections 3.2 and 3.3). `Python` has also been used to plot 2D graphs in order to analyze the performance of the developed algorithms (see section 3.3.2).

The visualization of the results is carried out by means of an external open-source tool called '`ParaView`' [16]. This software is a multi-platform data analysis and

visualization application well-known in the scientific community. It is able to show the results of the simulations by means of its Visualization Toolkit (VTK) library. In order to visualize the results, all we have to do is import the output file `.vtu` generated by our code (see section 3.3.3) into `ParaView`. Once in `ParaView`, there exist a wide range of options in order to customize the visualization and the rendering of the results in an interactive way, such as warping the 2-D scalar solution to the z dimension, coloring the scalar solution and displaying the solution of the flux as a vectorial field, among others.

3.2 Running the Finite Element codes

This thesis considers four different solvers: CG, HCG, DG and HDG. Figure 3.2 contains the specific flowchart of each one.

The definition of the problem statement (see section 3.2.1) is the first step in the codes. Once the problem is defined, two operations are required to get the solution of the problem: the building of the linear system (see section 3.2.2) and its resolution (see sections 3.2.3 and 3.2.4).

3.2.1 Problem Statement

The problem statement is defined in terms of:

- the mesh,
- the diffusivity tensor $\mathbf{D}(\boldsymbol{x})$,
- the source term $f(\boldsymbol{x})$,
- the Dirichlet boundary conditions $g_D(\boldsymbol{x})$,
- the Neumann boundary conditions $g_N(\boldsymbol{x})$,
- the τ parameter (only required in DG and HDG problems).

Additionally, some indicators can be selected in order to store information during the computation, such as the computational times, the memory consumptions and the errors (see further details in section 3.3.2).

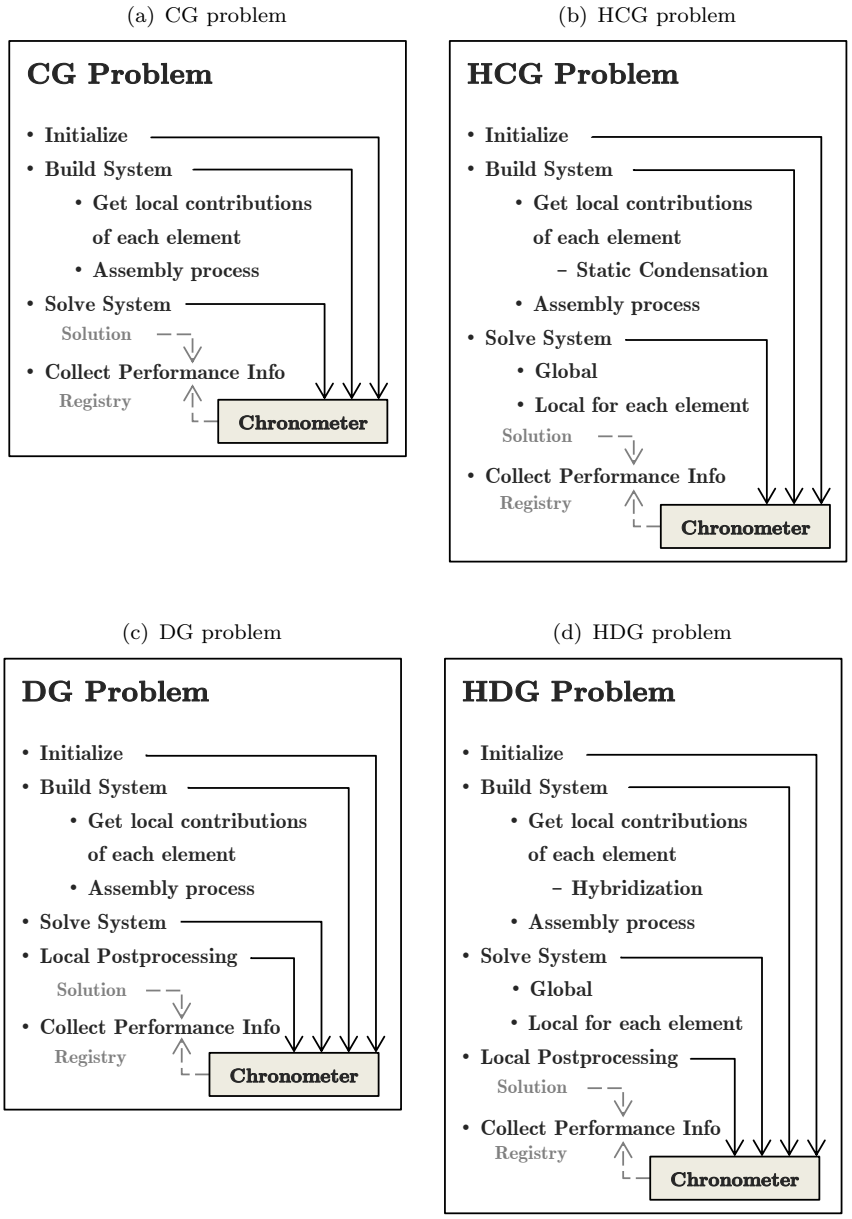


FIGURE 3.2: Flowcharts of the different problems.

3.2.2 Building the System

This section explains the steps to follow in order to build the linear system to be solved. Firstly, the elemental contributions have to be computed for each element. Additionally, for HCG and HDG problems, these contributions are hybridized. Once we get the proper elemental contributions, they are assembled into the global matrix and the global right hand side vector in order to get the linear system of equations.

Elemental contributions

The linear system is built from the assembly of the contributions of each element in the mesh. This section explains how to compute these elemental contributions. Both the elemental matrix $\mathbf{K}^e$ and the elemental right hand side vector $\boldsymbol{f}^e$ are implemented in `Python` as *dense* `Numpy` arrays, since they are relatively small and they are only used to perform the assembly process (see section 3.2.2-Assembly Process); this latter is the one which will consider *sparse* matrices.

The elemental contributions can be built in two ways:

- From an integral expression for each $[i, j]$ entry of it. This is the case of the continuous formulation, see equation (2.14a).
- From the concatenation of other submatrices. This is the case of the discontinuous formulation, see equations (2.51) and (2.48). Note that, in fact, these submatrices are built as well from integral expressions for each $[i, j]$ entry, as equation (2.46) shows.

In any case, the computation of integral expressions has to be carried out. The integration in the physical space is performed in the master element using numerical integration:

$$\int_{\Omega^e} f(\boldsymbol{x})\,\mathrm{d}\Omega = \int_{\Omega^M} \Big(f \circ \boldsymbol{\phi}\Big)(\boldsymbol{\xi}) \left|\frac{\partial \boldsymbol{\phi}(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right| \mathrm{d}\boldsymbol{\xi} \approx \sum_k w_k \left(f \circ \boldsymbol{\phi} \left|\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{\xi}}\right|\right)(\boldsymbol{\xi}_k), \tag{3.1}$$

where w_k and $\boldsymbol{\xi}_k$ are the integration weights and points, respectively, and $\boldsymbol{\phi}(\boldsymbol{\xi})$ is the mapping between the reference space and the physical space.

Once all the $\mathbf{K}^e[i, j]$ and $\boldsymbol{f}^e[i]$ are computed, we end up with two `Numpy` dense arrays for each element, $\mathbf{K}^e$ and $\boldsymbol{f}^e$. In the case of HCG and HDG problems, these elemental contributions are hybridized (see section 3.2.2-Hybridization). Finally, all the elemental contributions are properly assembled (see section 3.2.2-Assembly Process).

Hybridization

Each elemental contribution is hybridized in the case of HCG and HDG problems. In order to do so, we recall equations (2.24) and (2.26) from the HCG formulation (the same for the HDG case):

$$\mathbf{K}_H = \mathbf{K}_{11} - \mathbf{K}_{12} \cdot (\mathbf{K}_{22})^{-1} \cdot \mathbf{K}_{21}, \qquad \boldsymbol{f}_H = \boldsymbol{f}_1 - \mathbf{K}_{12} \cdot (\mathbf{K}_{22})^{-1} \cdot \boldsymbol{f}_2; \tag{3.2}$$

$$\boldsymbol{a}_2 = (\mathbf{K}_{22})^{-1} \cdot \boldsymbol{f}_2 - (\mathbf{K}_{22})^{-1} \cdot \mathbf{K}_{21} \cdot \boldsymbol{a}_1, \tag{3.3}$$

where the superindex e has been dropped for the sake of simplicity; label '1' accounts for the elemental primary unknowns and label '2' for the elemental secondary ones. Equation (3.2) is used together with (2.24) and (2.25) to find the primary unknowns $\boldsymbol{a}_1$, and equation (3.3) to find the secondary elemental unknowns $\boldsymbol{a}_2$ once the primary ones are already known. Both equations contain the inverse of the submatrix $\mathbf{K}_{22}$. In order to reduce the computational cost of the implementation, we propose the following procedure.

The first step is to define two *auxiliar arrays* $\mathbf{Z}$ and $\boldsymbol{z}$

$$\mathbf{Z} := (\mathbf{K}_{22})^{-1} \cdot \mathbf{K}_{21}, \tag{3.4a}$$

$$\boldsymbol{z} := (\mathbf{K}_{22})^{-1} \cdot \boldsymbol{f}_2, \tag{3.4b}$$

and to rewrite equations (3.2) and (3.3) using (3.4):

$$\mathbf{K}_H = \mathbf{K}_{11} - \mathbf{K}_{12} \cdot \mathbf{Z}, \qquad \boldsymbol{f}_H = \boldsymbol{f}_1 - \mathbf{K}_{12} \cdot \boldsymbol{z}; \tag{3.5}$$

$$\boldsymbol{a}_2 = \boldsymbol{z} - \mathbf{Z} \cdot \boldsymbol{a}_1, \tag{3.6}$$

The point now is to find the numerical expression of the auxiliar arrays $\mathbf{Z}$ and $\boldsymbol{z}$ from equations (3.4), avoiding the computation of $\mathbf{K}_{22}^{-1}$. To do so, we rewrite (3.4) as the following linear systems:

$$\mathbf{K}_{22} \cdot \mathbf{Z} = \mathbf{K}_{21}, \tag{3.7a}$$

$$\mathbf{K}_{22} \cdot \boldsymbol{z} = \boldsymbol{f}_2. \tag{3.7b}$$

Solving the linear systems in (3.7), the auxiliar arrays are found. Note that (3.7a) is, in fact, a set of n linear systems, where n is the number of columns of $\mathbf{K}_{21}$ or, equivalently, the number of primary unknowns. Together with (3.7b), the total number of linear systems to be solved is $n + 1$. It is worth, then, to apply a decomposition technique on the left hand side matrix $\mathbf{K}_{22}$, which is shared by all the linear systems to be solved. It has been considered the well-known *LU factorization* for $\mathbf{K}_{22}$

$$\mathbf{K}_{22} = \mathbf{L} \cdot \mathbf{U}, \tag{3.8}$$

for two main reasons: (*i*) it is already implemented in `Python` in an efficient way, and (*ii*) it is more general than the *Cholesky Factorization*, which is only valid for symmetric positive definite matrices. This is always the case in HCG schemes, but *not* in HDG ones.

Once $\mathbf{K}_{22}$ is factorized in $\mathbf{L}$ and $\mathbf{U}$, the `Python` LU solver gives us directly the auxiliar arrays $\mathbf{Z}$ and $\boldsymbol{z}$, which are used to find the hybridized local contributions of each element using equation (3.5), and proceed to its posterior assembly (see section 3.2.2-Assembly Process). In order to get also the secondary unknowns once the global system is solved (see section 3.2.3), the local solver has to be performed *element-by-element* retrieving equation (3.6), where the auxiliary arrays $\mathbf{Z}$ and $\boldsymbol{z}$ are needed again.

Assembly process

Up to this point, we have two `Python` arrays for each element, which correspond to their elemental contributions $\mathbf{K}^e$ and $\boldsymbol{f}^e$. Each elemental contribution, whether it has been hybridized (see section 3.2.2-Hybridization) or not, is assembled to conform the global system of equations to be solved.

Since we are developing an efficient code able to deal with very large problems, and the matrix of the global system is usually *sparse* in finite element schemes, the implementation in `Python` of the global system matrix has been done as well following two *sparse storage schemes* [29]:

Coordinate List (COO) Matrix which is very efficient for incremental building of sparse matrices (assembly process).

Compressed Sparse Row (CSR) Matrix which is very efficient to perform matrix operations (solving process).

The first sparse format, the COO, stores a list of (row, column, value) tuples. This specific format permits duplicate entries, and thus facilitates efficient construction of finite element matrices.

Once the Coordinate List matrix is built, it is transformed into the Compressed Sparse Row (CSR) format, which is the one that is used in order to solve the global system. This format is similar to COO, but compresses the row indices `I`. It is made up as well by three one-dimensional arrays, let us denote them by `AI`, `AJ` and `A`. The main idea of this format is to store the entries of the matrix in the array `A`, but this time *sorted* as they appear in a *row-wise* fashion. The entries equal to zero are ignored and the entries with duplicated (row,column) tuples are added. With these changes, the array `I` can be 'compressed' to become `AI`, which stores just the locations of the entries in `A` that *start* a row.

Next, we analyze the memory requirements of this implementation. We denote by nnz the number of non-zero entries of the matrix, and by n the number of rows or columns of

the original matrix. Instead of n^2 floating point entries as in the dense matrix approach, only $2 \cdot nnz + n + 1$ storage locations are required with the CSR approach, where nnz are floating point and $nnz + n + 1$ are integers (see further details in [13]). Typically, default `Python` floating point precision is double (`float64`), meaning that 64 bits (8 bytes) are needed to store a single float entry. Similarly, default `Python` integer entries are of the type `int32`, meaning that 32 bits (4 bytes) are needed to store a single integer [28]. , Therefore, the condition that has to be fulfilled in order to make the storage of a given matrix worthwhile in CSR format, instead of using the traditional dense format:

$$8 \cdot (nnz) + 4 \cdot (nnz + n + 1) \leq 8 \cdot \left(n^2\right), \tag{3.9}$$

which states that the total number of bytes required to store a given matrix in CSR format has to be less or equal than the total number of bytes required to store the same matrix in dense format. Thus, the maximum number of non-zero entries of a given matrix in order to make the CSR storage worthy is

$$nnz \leq \frac{2}{3} \cdot n^2 - \frac{1}{3} \cdot n - \frac{1}{3}. \tag{3.10}$$

According to the definition of the *sparsity index*, s_i (the number of non-zero entries of a matrix with respect to its total size), we find the maximum sparsity of a matrix in order to make its CSR storage worthy:

$$s_i = \frac{nnz}{n^2} \leq \frac{2}{3} - \frac{1}{3 \cdot n} - \frac{1}{3 \cdot n^2} \tag{3.11}$$

Although the equation (3.11) depends on n, it rapidly tends to just $2/3$ when we increase the matrix size, which is typically large enough in finite element computations. Therefore, we can state that *this CSR format implementation saves memory when the sparsity of the matrix to be stored is below 66%*, which means that only $1/3$ of the total entries in the matrix are required to be 0.

The usefulness of sparse matrices in terms of memory will be proved in chapter 4 once we check that the sparsity of our system matrices is lower than 66%, but we can advance that, for fine meshes or high interpolation degrees, the sparsity of the system matrices can reach values even below 1%.

Another advantage of CSR format is that it is very efficient in arithmetic matrix operations, requiring less CPU time than other sparse formats, and much less CPU time than the traditional dense approach [13]. Therefore, the usage of sparse matrices is highly recommended in efficient finite element codes.

The right hand side vector of the global system has to be assembled as well. It is implemented as a `numpy` floating point array, which is firstly initialized. Then, we check which entries of $\boldsymbol{f}^e$ are going to be assembled into the global right hand side vector, considering the source term, the Dirichlet boundary conditions and the Neumann boundary conditions. The global numbering of each local node is as well found, and then the assembly process takes place.

All the assembly process is carried out using a *vectorized approach*, which means that operations are performed *at once* on multiple scalars, rather than called several times on a single scalar. This technique saves a considerable amount of CPU time, specially for large arrays.

3.2.3 Solving the System

This thesis considers the use of *sparse iterative solvers* in order to get the solution of the global linear system. Incomplete LU factorization (ILU) is used to get the preconditioners of the linear system in order to speed up the convergence. In our case, this approach is preferred rather than other direct methods such as the LU factorization, which require the renumbering of the nodes prior to solving the system, in order to avoid large fill-in of the factorized matrices which would lead to unaffordable memory consumptions.

Thus, we use the *Conjugated Gradients (CG)* solver in the case of continuous problems (CG and HCG), since the system matrix is always symmetric and definite positive. Conversely, the *Generalized Minimal Residual (GMRES)* solver is considered for discontinuous problems (DG and HDG), where the system matrix is not symmetric. Both solvers are already implemented in `Python`. The *same* tolerance has been set for both solvers, in order to make the computations comparable.

Note that we get directly the whole solution if the problem is not hybridized (CG and DG). However, if the problem is hybridized (HCG and HDG) we also need to find the secondary unknowns of the original system using the local solver, explained in section 3.2.2-Hybridization.

3.2.4 Local Post-processing

The post-processing procedure can be implemented in mixed schemes (DG, HDG). Although the original solution is approximated using polynomials of degree (p), the post-processed solutions u and $\boldsymbol{q}$ are approximated using polynomial shape functions of degree $(p+1)$. This new shape functions cannot be taken from the original mesh, since we want

to increase the interpolation degree *just* for the solution, remaining with the same interpolation degree for the geometry. This procedure is well-known as *subparametric interpolation* [31].

The first step is to get the physical coordinates of the new nodes. This is straightforward, since the interpolation of the new geometry is *the same* as the one of the old geometry:

$$\boldsymbol{x}^e(\boldsymbol{\xi}) = \sum_{i=1}^{n_{old}} \phi_i(\boldsymbol{\xi}) \cdot \hat{\boldsymbol{x}}_i^e, \tag{3.12}$$

being n_{old} the number of shape functions of the old mesh, and $\hat{\boldsymbol{x}}_i^e$ the nodes of the old mesh. We just need to evaluate the old shape functions $\phi_i(\boldsymbol{\xi})$ at the nodes $\hat{\boldsymbol{\xi}}_{\text{new}}$ of the new master element to get the new physical coordinates $\hat{\boldsymbol{x}}_{\text{new}}^e$:

$$[\hat{\boldsymbol{x}}_{\text{new}}^e]_j = \boldsymbol{x}^e([\hat{\boldsymbol{\xi}}_{\text{new}}]_j) = \sum_{i=1}^{n_{old}} \phi_i([\hat{\boldsymbol{\xi}}_{\text{new}}]_j) \cdot \hat{\boldsymbol{x}}_i^e, \qquad \forall j = 1 \dots n_{new}, \tag{3.13}$$

being n_{new} the number of shape functions of the new geometry. Note that at this point the physical geometry of the new element is already defined.

Next step is to map the current solutions $u(\boldsymbol{x})$ and $\boldsymbol{q}(\boldsymbol{x})$ to the nodes of the new mesh to get $\hat{u}_{\text{new}}^e$ and $\hat{\boldsymbol{q}}_{\text{new}}^e$, with a double purpose: (i) to be able to build the system for the post-process, and (ii) to export a single mesh (the post-processed one) containing all the solutions, the post-processed one $u_{\text{post}}(\boldsymbol{x})$ and the previous ones $u(\boldsymbol{x})$ and $\boldsymbol{q}(\boldsymbol{x})$. This procedure is performed in the same way that we have used to interpolate the geometry; that is, the values of the old solution are interpolated within the e-th element using the old shape functions:

$$[\hat{u}_{\text{new}}^e]_j = \sum_{i=1}^{n_{old}} \phi_i([\hat{\boldsymbol{\xi}}_{\text{new}}]_j) \cdot \hat{u}_i^e, \qquad \forall j = 1 \dots n_{new}; \tag{3.14a}$$

$$[\hat{\boldsymbol{q}}_{\text{new}}^e]_j = \sum_{i=1}^{n_{old}} \phi_i([\hat{\boldsymbol{\xi}}_{\text{new}}]_j) \cdot \hat{\boldsymbol{q}}_i^e, \qquad \forall j = 1 \dots n_{new}. \tag{3.14b}$$

We can now build the linear system corresponding to the post-process (see equation (2.54)) discretizing u_{post}^e, $\boldsymbol{q}^e$ and the test functions w^e in the new mesh using the new shape functions of polynomial degree $(p+1)$, $\phi_{\text{new}\,j}^e(\boldsymbol{x})$, and the nodal values computed

in eq. (3.14):

$$u^e_{\text{post}} \approx \sum_{j=1}^{n_{new}} [\phi^e_{\text{new}}]_j(\boldsymbol{x}) \cdot [\hat{u}^e_{\text{post}}]_j; \tag{3.15a}$$

$$\boldsymbol{q}^e \approx \sum_{j=1}^{n_{new}} [\phi^e_{\text{new}}]_j(\boldsymbol{x}) \cdot [\hat{\boldsymbol{q}}^e_{\text{new}}]_j; \tag{3.15b}$$

$$w^e := [\phi^e_{\text{new}}]_i(\boldsymbol{x}), \qquad \forall i = 1 \dots n_{new}, \tag{3.15c}$$

as well as the gradient of u^e_{post}:

$$\nabla u^e_{\text{post}} \approx \sum_{j=1}^{n_{new}} [\nabla \phi^e_{\text{new}}]_j(\boldsymbol{x}) \cdot [\hat{u}^e_{\text{post}}]_j, \tag{3.16}$$

to get

$$\int_{\Omega^e} [\nabla \phi^e_{\text{new}}]_i \cdot [\nabla \phi^e_{\text{new}}]_j \, \mathrm{d}\boldsymbol{x} \cdot [\hat{u}^e_{\text{post}}]_j = -\int_{\Omega^e} [\nabla \phi^e_{\text{new}}]_i \cdot \mathbf{D}^{-1} \cdot [\phi^e_{\text{new}}]_j \cdot [\hat{\boldsymbol{q}}^e_{\text{new}}]_j \, \mathrm{d}\boldsymbol{x}, \quad \forall i = 1 \dots n_{new}. \tag{3.17}$$

The matrix form of (3.17) is then:

$$\mathbf{K}^e \cdot \hat{u}^e_{\text{post}} = \boldsymbol{f}^e \tag{3.18}$$

if we make the following definitions:

$$\mathbf{K}^e[i,j] = \int_{\Omega^e} [\nabla \phi^e_{\text{new}}]_i \cdot [\nabla \phi^e_{\text{new}}]_j \, \mathrm{d}\boldsymbol{x}; \tag{3.19a}$$

$$\boldsymbol{f}^e[i] = -\int_{\Omega^e} [\nabla \phi^e_{\text{new}}]_i \cdot \mathbf{D}^{-1} \cdot [\phi^e_{\text{new}}]_j \cdot [\hat{\boldsymbol{q}}^e_{\text{new}}]_j \, \mathrm{d}\boldsymbol{x}. \tag{3.19b}$$

The linear system in (3.18) is singular and requires one equation to be substituted by the discretization of the equation seen in (2.55):

$$\int_{\Omega^e} [\phi^e_{\text{new}}]_j \, \mathrm{d}\boldsymbol{x} \cdot [\hat{u}^e_{\text{post}}]_j = \int_{\Omega^e} \phi^e_i \hat{u}^e_i \, \mathrm{d}\boldsymbol{x}. \tag{3.20}$$

The linear system is now well-posed and invertible, and it can be solved using numerical integration (see section 3.2.2-Elemental Contributions) in order to get $\hat{u}^e_{\text{post}}$. The post-processed solution u^e_{post} within the e-th element is retrieved using eq. (3.15a). This procedure is repeated for all the elements in order to get u_{post}.

3.3 After the Simulation

Once the problem has been solved, two kinds of information are obtained. On the one hand, we get `Python` arrays which contain the solution of the problem. In the CG and HCG cases, just a single array is obtained, corresponding to the nodal values of the scalar unknown $u(\boldsymbol{x})$; in the DG and HDG cases, the nodal values of the flux $\boldsymbol{q}(\boldsymbol{x})$ and the post-processed solution $u_{\text{post}}(\boldsymbol{x})$ are additionally obtained. Section 3.3.1 details how to compute the error of these numerical solutions with respect to the analytical ones. On the other hand, we also get the registry of the computation, explained in section 3.3.2. This will allow us to measure the computational requirements and the accuracy of the analyzed methods.

3.3.1 Error Computation

To assess the accuracy of the results obtained by our implementation, we will use L_2-error. The L_2-error of a function $\boldsymbol{f}(\boldsymbol{x})$ with respect to another function $\boldsymbol{g}(\boldsymbol{x})$ is defined as the L_2-norm of the residual vector $\boldsymbol{r}(\boldsymbol{x})$, where $\boldsymbol{r}(\boldsymbol{x}) := \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{x})$:

$$\|\boldsymbol{r}(\boldsymbol{x})\|_{L_2(\Omega)} := \left(\int_{\Omega} \boldsymbol{r}(\boldsymbol{x}) \cdot \boldsymbol{r}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \right)^{1/2}. \tag{3.21}$$

Therefore, our aim is to compute:

- The L_2-error of the scalar unknown $u(\boldsymbol{x})$ with respect to $u_{\text{an}}(\boldsymbol{x})$.
- The L_2-error of the flux $\boldsymbol{q}(\boldsymbol{x})$ with respect to $\boldsymbol{q}_{\text{an}}(\boldsymbol{x}) = -\mathbf{D} \cdot \nabla u_{\text{an}}(\boldsymbol{x})$.
- The L_2-error of the scalar unknown $u_{\text{post}}(\boldsymbol{x})$ with respect to $u_{\text{an}}(\boldsymbol{x})$.

Note that in continuous schemes (CG and HCG) the only available solution is $u(\boldsymbol{x})$, although $\boldsymbol{q}(\boldsymbol{x})$ can be derived from $u(\boldsymbol{x})$ since we have that $\boldsymbol{q}(\boldsymbol{x}) = -\mathbf{D} \cdot \nabla u(\boldsymbol{x})$. However, $u_{\text{post}}(\boldsymbol{x})$ is not available. Therefore, in these two cases we will compute only the L_2-errors of $u(\boldsymbol{x})$ and $\boldsymbol{q}(\boldsymbol{x})$. For the other two schemes (DG and HDG) we can compute the three previous errors directly.

In any case, the solutions are defined locally within every element, since its shape functions are defined so. Therefore, the computation of the L_2-errors will be performed element by element:

$$\|\boldsymbol{r}(\boldsymbol{x})\|_{L_2(\Omega)} := \left(\sum_{\Omega^e \in \Omega} \int_{\Omega^e} \boldsymbol{r}^e(\boldsymbol{x}) \cdot \boldsymbol{r}^e(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \right)^{1/2}, \tag{3.22}$$

where $\boldsymbol{r}^e(\boldsymbol{x})$ depends on the problem we are solving (continuous or discontinuous) in the following way:

$$u(\boldsymbol{x}) \text{ for all problems :} \qquad \boldsymbol{r}^e = \sum_j \phi_j^e \cdot \hat{\boldsymbol{u}}^e[j] - u_{\text{an}}; \tag{3.23a}$$

$$\boldsymbol{q}(\boldsymbol{x}) \text{ for CG and HCG :} \qquad \boldsymbol{r}^e[k] = \sum_l \left[\sum_j \left(-\mathbf{D}_{kl} \frac{\partial \phi_j^e}{\partial x_l} \cdot \hat{\boldsymbol{u}}^e[j] \right) + \mathbf{D}_{kl} \frac{\partial u_{\text{an}}}{\partial x_l} \right], \quad \forall k = 1 \dots n_{sd}; \tag{3.23b}$$

$$\boldsymbol{q}(\boldsymbol{x}) \text{ for DG and HDG :} \qquad \boldsymbol{r}^e[k] = \sum_j \phi_j^e \cdot \hat{\boldsymbol{q}_{\boldsymbol{k}}}^e[j] + \sum_l \mathbf{D}_{kl} \frac{\partial u_{\text{an}}}{\partial x_l}, \quad \forall k = 1 \dots n_{sd}; \tag{3.23c}$$

$$u_{\text{post}}(\boldsymbol{x}) \text{ for DG and HDG :} \qquad \boldsymbol{r}^e = \sum_j {\phi_{\text{post}}}_j^e \cdot \hat{\boldsymbol{u}}_{\text{post}}{}^e[j] - u_{\text{an}}. \tag{3.23d}$$

The integrals in equation (3.22) will be computed numerically as usual (see section 3.2.2-Elemental Contributions).

3.3.2 Registry

The registry of the problem contains a certain number of performance indicators, that are useful to generate line plots of the quantity of interest over the parameter involved in the analysis (typically the mesh size h or the polynomial degree p) by means of the `Python` plotters.

Time indicators. We have divided our implementation in several sections, and we have assigned a time indicator for each one of them. For CG and HCG problems, we have used three temporal indicators: the time to *initialize* the problem, the time to *build* the global system of equations (including the computation of the elemental contributions and the assembly process) and the time to *solve* it. For DG and HDG problems, we have included an additional indicator, which stores the time to perform all the *postprocess* operations in order to obtain $u_{\text{post}}(\boldsymbol{x})$. Note that, for hybridized problems (HCG and HDG), the *build* time refers as well to the time required to *hybridize* (or similarly condensate) the local systems for each element. Similarly, the *solve* time refers to both the global solver and the local solvers of each element.

Time indicators are used to plot time graphs, where the x-axis represents the element size, h, and the y-axis represents the ratio between two times of computation, typically HDG over HCG.

Memory indicators. Each computation has a cost in terms of memory consumption. This cost is not negligible and might be the limiting factor when running large problems, depending on the available computational memory. The memory consumption peak is associated to the size of the system matrix. For this reason the memory indicator contains the size (referring to bytes) of the *system matrix* to be solved. For hybridized problems (HCG, HDG), the size of the *auxiliar array* (see section 3.2.2-Hybridization) is stored as well.

All this information can be plotted in the memory plots, where the x-axis represents the element size, h, and the y-axis represents the memory required to store the system matrix in KB.

Sparsity indicators. This indicator quantifies the sparsity of the matrix system in terms of the sparsity indicator, s_i, the ratio between the *number of non-zero entries* of it and the *total number of entries*. The x-axis in the sparsity graphs represents the element size, h, and the y-axis the sparsity of the system matrix in percentage.

Unknowns indicators. The information about the number of *primary unknowns* and the number of *secondary unknowns* of the global system is stored here. Note that in the case of CG and DG problems, since they are not hybridized, the former will be equal to the total number of unknowns and the latter will be equal to zero.

This information can be plotted in the unknowns plots, where the x-axis represents the element size, h, and the y-axis represents the number of unknowns of the system.

Error indicators. This indicators have been computed according to section 3.3.1. The CG and HCG problems store the error of the scalar unknown u and the flux unknown $\boldsymbol{q}$; the DG and HDG problems store as well the error of the post-processed solution u_{post}.

The error plots can be built using this indicators. The x-axis of this kind of graphs represents the element size, h; it is typically set in log_{10}-scale. The y-axis represents the L_2-error of the numerical solution with respect to the analytical one in a log_{10}-scale. Since the L_2-error theoretically converges at a constant rate of r for a given polynomial interpolation degree p and for a given solver (see table 3.1), straight lines with a slope of r are expected for each kind of variable and problem.

Note that not all the previous indicators may be filled in; only the ones that have been selected in the problem statement (see section 3.2.1) may contain information. All the rest are empty variables.

	CG and HCG	DG and HDG
$u_{\text{post}}(\boldsymbol{x})$	-	$p+2$
$u(\boldsymbol{x})$	$p+1$	$p+1$
$\boldsymbol{q}(\boldsymbol{x})$	p	$p+1$

TABLE 3.1: Theoretical convergence rate r of the L_2-error of the solutions.

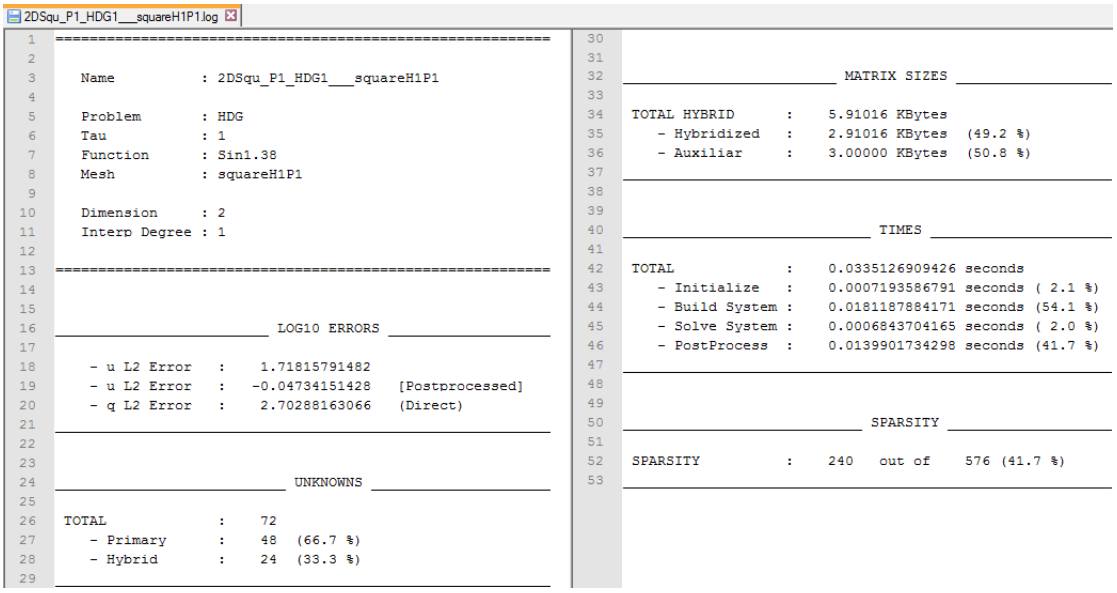


FIGURE 3.3: Example of `.log` registry file: `2DSqu_P1_HDG1___squareH1P1`.

3.3.3 Exporting the Results

Both the solution of the computation and its registry are exported in order to store them *outside* `Python`. In order to do that, the *writer* functions are required.

Two `Python` writers are implemented in this thesis: the `VTKWriter`, which exports the solution in a `.vtu` file, and the `logWriter`, which exports the registry in a `.log` file. Both are `ASCII` files.

The `VTKWriter` was already implemented by my mentors when I first received the code, so *it's not part of my own work*. It writes the solution in a way that `Paraview` is able to read (see section 3.1-Visualization of the results).

The `logWriter` was totally implemented by me. Let's take a look at the registry file in figure 3.3. It consists in an HDG computation with $\tau = 1$ in the mesh `SquareH1P1.dcm`. This file corresponds to a 2-D structured quadrilateral mesh for the $[0,1] \times [0,1]$ domain which has 2 linear squares in each x-y direction. The registry file is named `2DSqu_P1_HDG1___squareH1P1.log`, according to the `Analysis Name` (`2DSqu_P1`), the `Problem Name` (`HDG1` including the τ parameter for discontinuous problems) and the mesh name (`squareH1P1`).

The header lines (1 to 13) contain general information about the computation: the name of the registry file, the name of the solver, the value of the τ parameter (only for discontinuous solvers), the identifier of the analytical problem (here, `Sin1.38` refers to a sinusoidal function in both $x-$ and $y-$ directions with a wave frequency of 1.38), the name of the mesh file, the dimension of the problem and the interpolation degree of the shape functions.

The rest of the file is composed of 5 sections, each one corresponding to a kind of performance indicator (see section 3.3.2). Note that the errors are always written in $\log_{10}$-scale. The L_2-error of the scalar solution is shown in both its original and post-processed form for discontinuous problems. The flux L_2-error has a flag indicating whether it has been computed using equation (3.23b) `(Indirect)` or (3.23c) `(Direct)`. The rest of the file is easily readable and presents no further particularities.

Chapter 4

Results and Validation

The aim of this chapter is to validate the capabilities and the performance of the developed code. To this end, we show that it can handle several kinds of geometries (even with curved boundaries) in 2D and 3D, different interpolation degrees and mesh sizes, and domains with several source terms and diffusion tensors, with both Dirichlet and Neumann boundary conditions. A comparison between the four implemented solvers (CG, HCG, DG and HDG) is performed in terms of the L_2-error with respect to the analytical solution, as well as the time and memory requirements for each simulation. Additionally, the number of unknowns of each global system and the sparsity of the matrix system are analyzed. All the computations have been carried out at the cluster `Clonetroop` of the *Laboratori de Càlcul Numèric* (LaCàN) of the UPC [18].

This chapter is organized as follows: section 4.1 contains several studies of convergence of the L_2-error of the solution in simple domains, in order to prove the reliability and accuracy of the codes. Section 4.2 compares the performance of the different computations presented in section 4.1 in terms of CPU time and memory consumptions. In addition, we also analyze the number of unknowns of the global systems and its sparsity. Section 4.3 shows the reliability of the different solvers when complex geometries with curved boundaries are considered. Finally, section 4.4 shows the effect of considering non-homogeneous diffusion tensors in the domain.

4.1 Error Convergence

In this section, two domains are considered:

- A square $[0,1] \times [0,1]$ domain for 2D cases. Dirichlet boundary conditions are prescribed at the edges $x = 0$ and $x = 1$, and Neumann boundary conditions are prescribed at the edges $y = 0$ and $y = 1$.
- A cubic $[0,1] \times [0,1] \times [0,1]$ domain for 3D cases. Dirichlet boundary conditions are prescribed at the faces $x = 0$, $y = 0$ and $z = 0$, and Neumann boundary conditions are set at $x = 1$, $y = 1$ and $z = 1$.

We consider the stationary diffusion problem:

$$\nabla(\mathbf{D} \cdot \nabla u) = -f,$$

where $\mathbf{D} := \mathbf{I}_{n_{sd}}$ in order to get an homogeneous domain. Dirichlet and Neumann boundary conditions, as well as the source term f, have been properly selected in order to give an exact solution of the form

$$u(x,y) = \sin(2\pi kx) \cdot \sin(2\pi ky) \qquad \text{for the 2D case,} \tag{4.1a}$$

$$u(x,y,z) = \sin(2\pi kx) \cdot \sin(2\pi ky) \cdot \sin(2\pi kz) \qquad \text{for the 3D case,} \tag{4.1b}$$

with $k = 1.38$.

In order to solve this problem, three kind of meshes have been considered:

- Structured quadrilateral meshes for 2D domains.
- Unstructured triangular meshes for 2D domains.
- Structured hexahedral meshes for 3D domains.

All the considered meshes are present in appendix A.

The length of the edges of the mesh elements is set to h, in a way that each edge along the boundary of the domain is divided in $1/h$ segments. The polynomial degree of the shape functions of each element is denoted with p.

The convergence of the L_2-error of the solution of this problem is analyzed in two different ways. In section 4.1.1, the polynomial degree p is fixed whereas the mesh is refined. With this procedure, the degrees of freedom of the global system increase, and the

error is expected to decrease according to table 3.1. Section 4.1.2 introduces a different approach: the idea is to generate meshes with a constant h/p ratio; in this way, all these meshes lead to the same number of mesh nodes, and thus to the same spatial resolution. Then, we are able to compare two refinement approaches: the h-refinement approach (increasing the number of elements in the mesh) and the p-refinement (increasing the degree of the polynomial shape functions).

4.1.1 Convergence Rates for each Polynomial Degree p

This section considers the meshes generated with an element size of $h = 2^{-s}$ where $s = 1 \dots 7$, and with a polynomial degree of $p = 1 \dots 5$. In this way, we end up with 35 meshes for each kind of discretization (quadrilaterals, triangles and hexahedrals). These meshes are used to perform the simulations in this section; however, due to memory limitations, not all of them have been finally used in all the cases.

Note that the hybridization of the linear systems has a minimum impact on the total error. In fact, a given linear system and its hybridized version are both originally built from the same formulation, discretization and numerical integration. Thus, they lead to the *same solution* if we neglect round-off errors. For this reason, next lines consider either CG or HCG indistinctly as HCG, and DG and HDG are treated as HDG.

Figures 4.1, 4.2 and 4.3 show the L_2-error of the numerical solutions of the problem stated in section 4.1 with respect to their corresponding analytical solutions using quadrilateral, triangular and hexahedral meshes, respectively. Each subplot in these figures refers to a different solution (u, u_{post} and $\boldsymbol{q}$) and solver (HCG and HDG). The numbers over the curves correspond to the slope of each one of them in a $(\log_{10} - \log_{10})$ scale; these slopes coincide with the theoretical convergence rates shown in table 3.1. The scalar unknown u theoretically converges at a rate of $(p+1)$, and the post-processed solution u_{post} at a rate of $(p+2)$. Similarly, the flux unknown $\boldsymbol{q}$ theoretically converges at a rate of (p) in the HCG case and at a rate of $(p+1)$ in the HDG case.

From these three figures, we realize that our numerical results converge with the theoretical convergence rates when the convergence region is reached. For continuous problems, the theoretical convergence rates are rapidly achieved, whereas for discontinuous problems this fact may slightly depend of the τ parameter, which controls the element-wise discontinuity of the scalar solution u and the normal component of the flux $\boldsymbol{q}$.

The post-processed solution u_{post} provides more accurate results than just u without post-process, because the interpolation degree of the former $(p+1)$ is higher than the one of the latter (p). Since the post-processed solution u_{post} is obtained from the flux

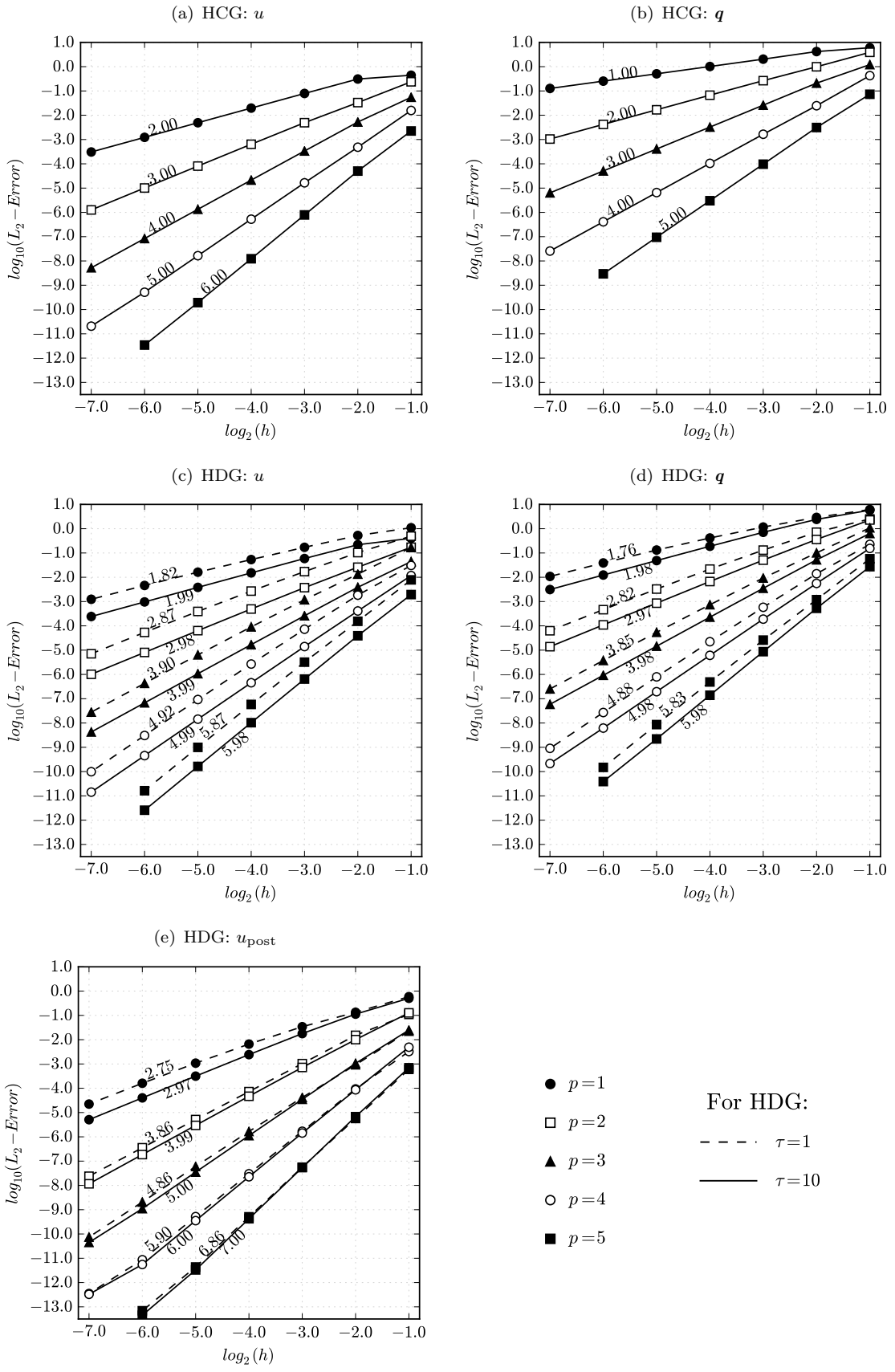


FIGURE 4.1: L_2-error convergence on structured quadrilateral meshes.

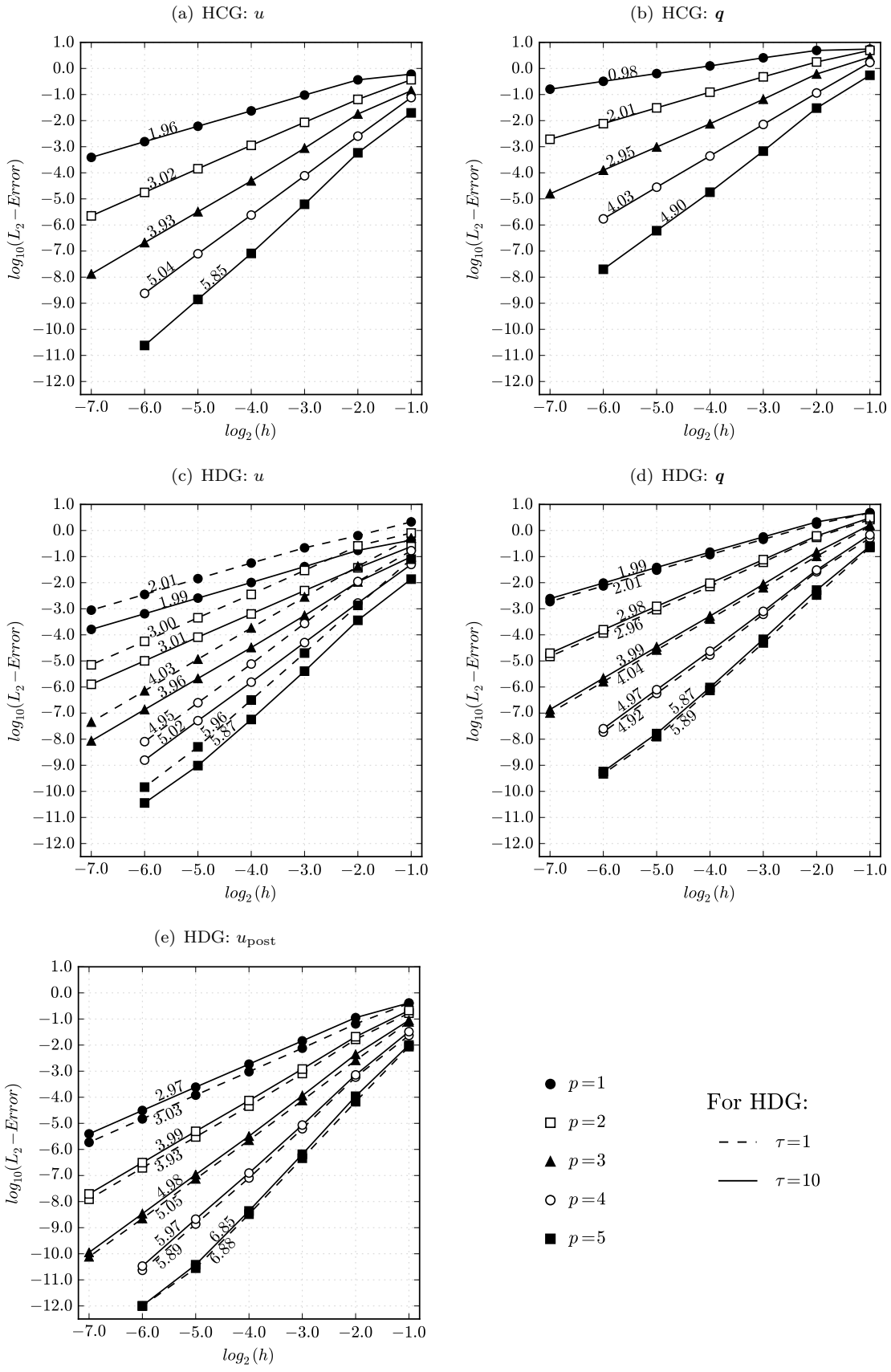


FIGURE 4.2: L_2-error convergence on unstructured triangular meshes.

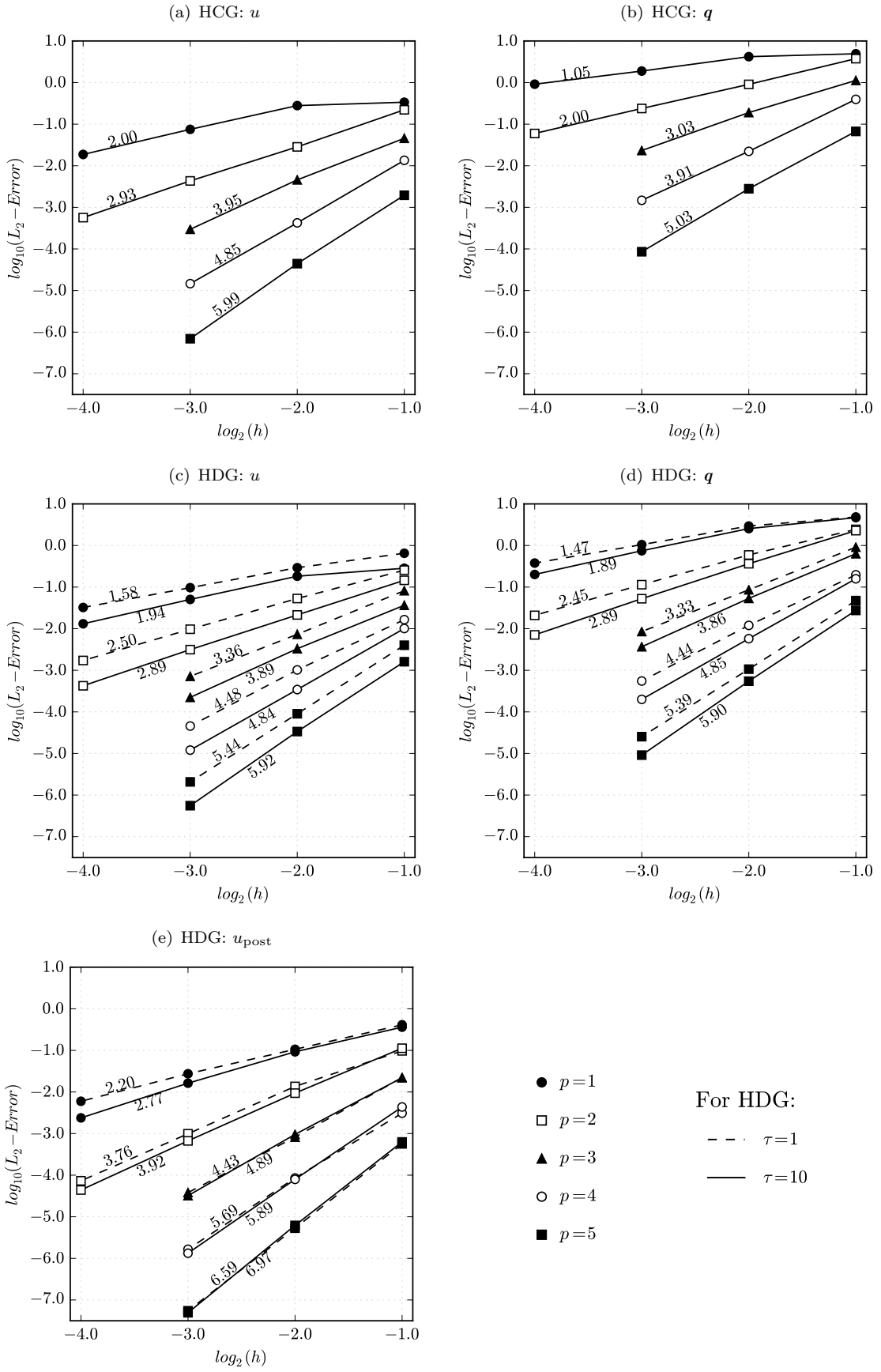


FIGURE 4.3: L_2-error convergence on structured hexahedral meshes.

$\boldsymbol{q}$, the accuracy gain of the scalar unknown when using local post-processing depends on the inherent error of $\boldsymbol{q}$. To illustrate this affirmation note that, for all kind of meshes, $\tau = 10$ provides more accurate results for u than $\tau = 1$. For quadrilaterals and hexahedral meshes, the flux $\boldsymbol{q}$ is also more accurate with $\tau = 10$, and thus the post-processed solution u_{post} leads to less error when $\tau = 10$ is considered instead of $\tau = 1$. Conversely, for triangular meshes, it comes out that $\tau = 1$ leads to a more accurate solution for the flux $\boldsymbol{q}$, and for this reason the post-processed scalar solution u_{post} is also better approximated with $\tau = 1$ in this case.

These numerical results agree with the theoretical convergence rates. Thus, we checked that our code provides the correct solution with the expected convergence rates for the L_2-error when quadrilateral, triangular and hexahedral meshes are considered.

4.1.2 High-Order meshes with Constant Spatial Resolution

There exist two different approaches in order to reduce the norm of the error in finite element simulations: a first approach is to refine the mesh (decrease h); a different approach is to increase the polynomial degree of the shape functions used to interpolate the solution (increase p). Since the error of the scalar solution u is of order $\mathcal{O}(h^{p+1})$, it seems to be more interesting to increase p rather than decrease h. This section considers 4 quadrilateral meshes with a constant h/p ratio:

a) 2D structured quadrilateral mesh with $h = 1/16$ and $p = 1$

b) 2D structured quadrilateral mesh with $h = 1/8$ and $p = 2$

c) 2D structured quadrilateral mesh with $h = 1/4$ and $p = 4$

d) 2D structured quadrilateral mesh with $h = 1/2$ and $p = 8$

These meshes are presented in appendix A.

Despite the fact that these meshes are refined in a different way, all of them consist of 289 nodes, 17 for each x, y direction. Therefore, all meshes have the same spatial resolution.

For CG simulations, this means that all of them will lead to 289 scalar unknowns (without taking into account the Dirichlet boundary conditions). When HCG is considered, the number of unknowns of the global linear system decreases from mesh *a*) to mesh *d*) since the interior nodes are condensed, leading to more efficient computations.

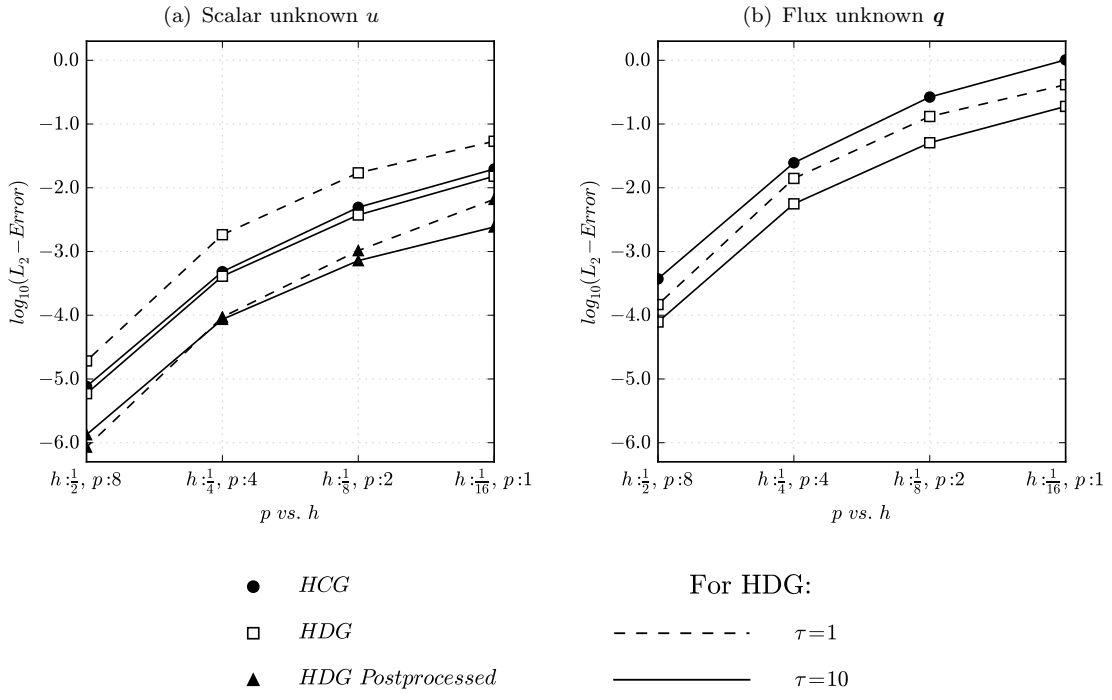


FIGURE 4.4: L_2-error convergence of the solution on quadrilateral meshes with constant h/p ratio.

	CG	**HCG**	**DG**	**HDG**
Mesh a) : $h = 1/16,\ p = 1$	255	255	4160	1088
Mesh b) : $h = 1/8,\ \ p = 2$	255	191	2160	432
Mesh c) : $h = 1/4,\ \ p = 4$	255	111	1400	200
Mesh d) : $h = 1/2,\ \ p = 8$	255	59	1080	108

TABLE 4.1: Degrees of freedom of the global system for each solver and mesh.

	CG	**HCG**	**DG**	**HDG**
Mesh a) : $h = 1/16,\ p = 1$	$1.95 \cdot 10^{-2}$		$2.40 \cdot 10^{-3}$	
Mesh b) : $h = 1/8,\ \ p = 2$	$4.90 \cdot 10^{-3}$		$7.24 \cdot 10^{-4}$	
Mesh c) : $h = 1/4,\ \ p = 4$	$4.79 \cdot 10^{-4}$		$8.71 \cdot 10^{-5}$	
Mesh d) : $h = 1/2,\ \ p = 8$	$7.59 \cdot 10^{-6}$		$1.35 \cdot 10^{-6}$	

TABLE 4.2: L_2-error of the scalar unknown u for each solver and mesh. Discontinuous problems show the post-processed solution with $\tau = 10$.

For DG simulations, the number of unknowns is reduced from mesh a) to mesh d) even with no hybridization, since at each mesh there exist less nodes on elemental boundaries. Every single node in DG can correspond to several scalar unknowns (one unknown for each adjacent element and another one for each concurrent numerical trace) and to several flux unknowns (one unknown for each adjacent element and dimension). When hybridization is considered, this reduction is even more remarkable.

In any case, moving from fine meshes with low interpolation degrees, such as mesh a),

towards coarse meshes with high interpolation degrees, such as mesh d), supposes a reduction of the number of unknowns of the global system (see table 4.1).

On top of that, the L_2-error of the solution is reduced as well, as table 4.2 and figure 4.4 show. Both the scalar unknown, u, with or without post-process, and the flux unknown, $\boldsymbol{q}$, are more accurate when coarse high-order meshes are considered instead of fine low-order ones. For instance, the L_2-error of the post-processed scalar unknown u_{post} decreases *three* orders of magnitude considering high-order meshes, from $2.40 \cdot 10^{-3}$ to $1.35 \cdot 10^{-6}$.

Therefore, we have shown that high order methods are more accurate than linear methods using the same spatial resolution.

4.2 Performance Analysis

The performance of the computations in 4.1.1 is assessed in this section in terms of memory consumption (section 4.2.1) and computational times (section 4.2.2). The rank of the global system and the sparsity of the global matrices are analyzed in sections 4.2.3 and 4.2.4 respectively.

The performance of every single simulation is covered in appendix B. Instead, we focus here on a) the effect of the hybridization technique on the main performance indicators, and b) the comparison between continuous and discontinuous approaches. For this reason, we show *relative* data in this section.

Just hybrid schemes are considered for hexahedral meshes, since they are very memory- and time-demanding. Therefore, the effect of hybridization will be only assessed on triangular and quadrilateral meshes.

Note that the τ parameter does *not* have any effect in terms of memory requirements, and has a negligible effect on the computational times. Therefore, a single value $\tau = 10$ has been considered for all the discontinuous computations of this section.

4.2.1 Memory Consumption

Figure 4.5 shows the ratio between the hybrid and non-hybrid solvers in terms of memory consumptions using structured quadrilateral meshes and unstructured triangular meshes, depending on the polynomial degree p and on the mesh element size h. The *total* memory requirements shown here refer in hybrid cases to the memory needed to store *both* the global system matrix and the auxiliary arrays $\mathbf{Z}^e$ and $\mathbf{z}^e$ for each element.

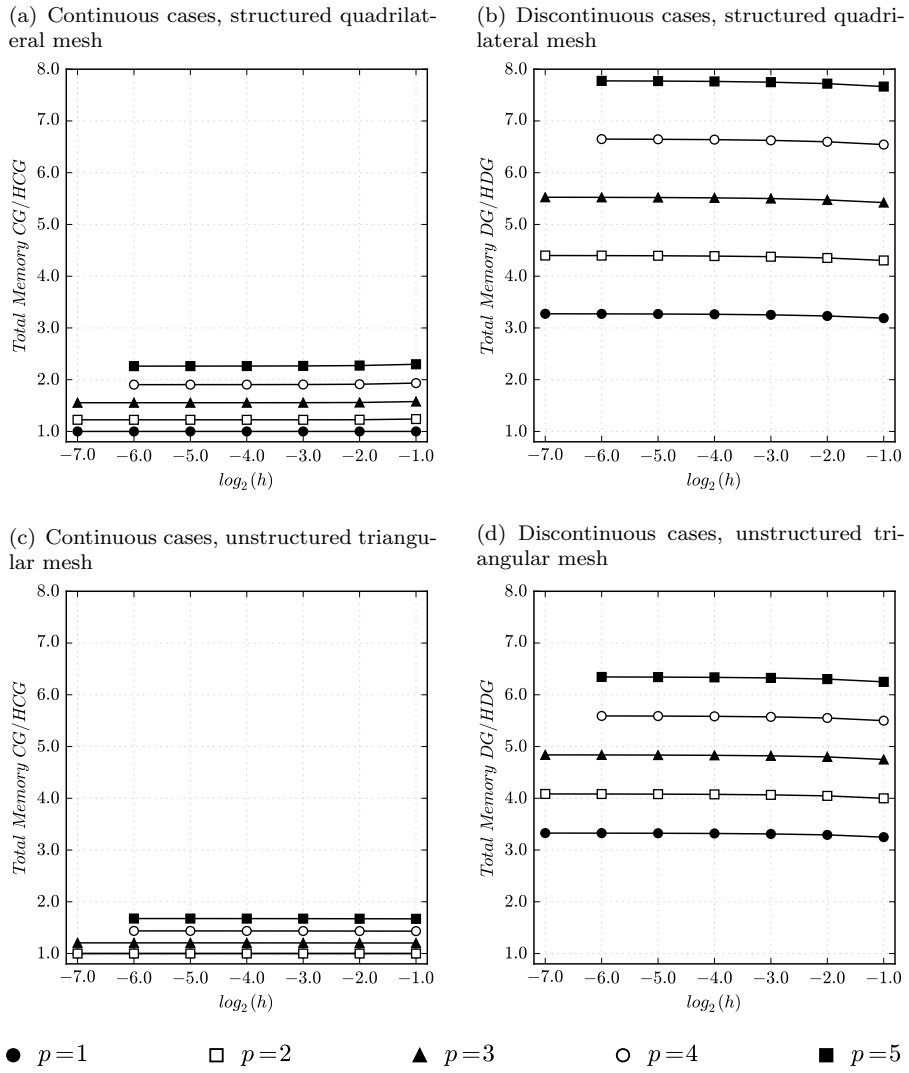


FIGURE 4.5: Impact of hybridization on the memory consumption.

From figure 4.5, it is clear that the hybridization technique saves memory in any case, but specially when high-order meshes are considered. For discontinuous solvers, the memory saving is much more remarkable than for the continuous ones. For instance, for an interpolation degree of $p = 5$, the memory consumption in the discontinuous approach gets reduced because of hybridization by a factor of 8 on quadrilateral meshes, and by a factor of 6 on triangular meshes. For the continuous approach, this factor is around 2 on both kind of meshes. So, in any case, hybridization saves memory. We note that this memory saving due to hybridization is almost independent of the element size h.

We compare continuous and discontinuous approaches now. The memory consumption ratios between hybridized continuous and discontinuous solvers appear in figure 4.6. Discontinuous schemes appear to require more memory than continuous ones, either if

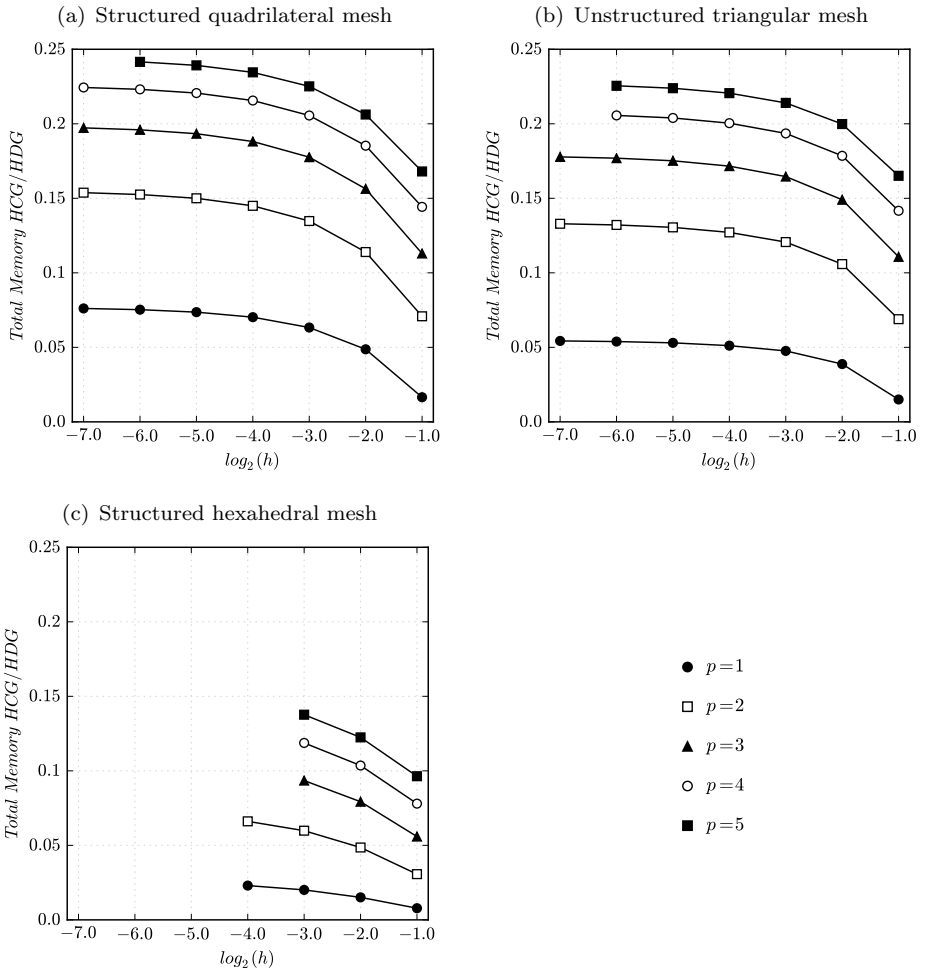


FIGURE 4.6: Comparison of memory consumption between HCG and HDG solvers.

hybridization is considered or not. It makes sense, since every single node in the continuous approach holds just a single scalar unknown. This is not the case in discontinuous schemes, where the nodes along the elemental boundaries may hold more than one scalar unknown for the different concurrent numerical traces. We note that, for coarse meshes and low interpolation degrees, HDG spends much more memory than HCG, but this difference gets reduced when high-order fine meshes are considered. For instance, if we consider a fine quadrilateral mesh ($h < 1/32$) and an interpolation degree of $p = 3$, the HDG spends 5 times more memory than HCG. However, for $p = 5$ this ratio is reduced to 4. On triangular meshes, HDG is slightly more memory demanding than previously, and when we move to hexahedral meshes, the relative memory-consumption is even larger.

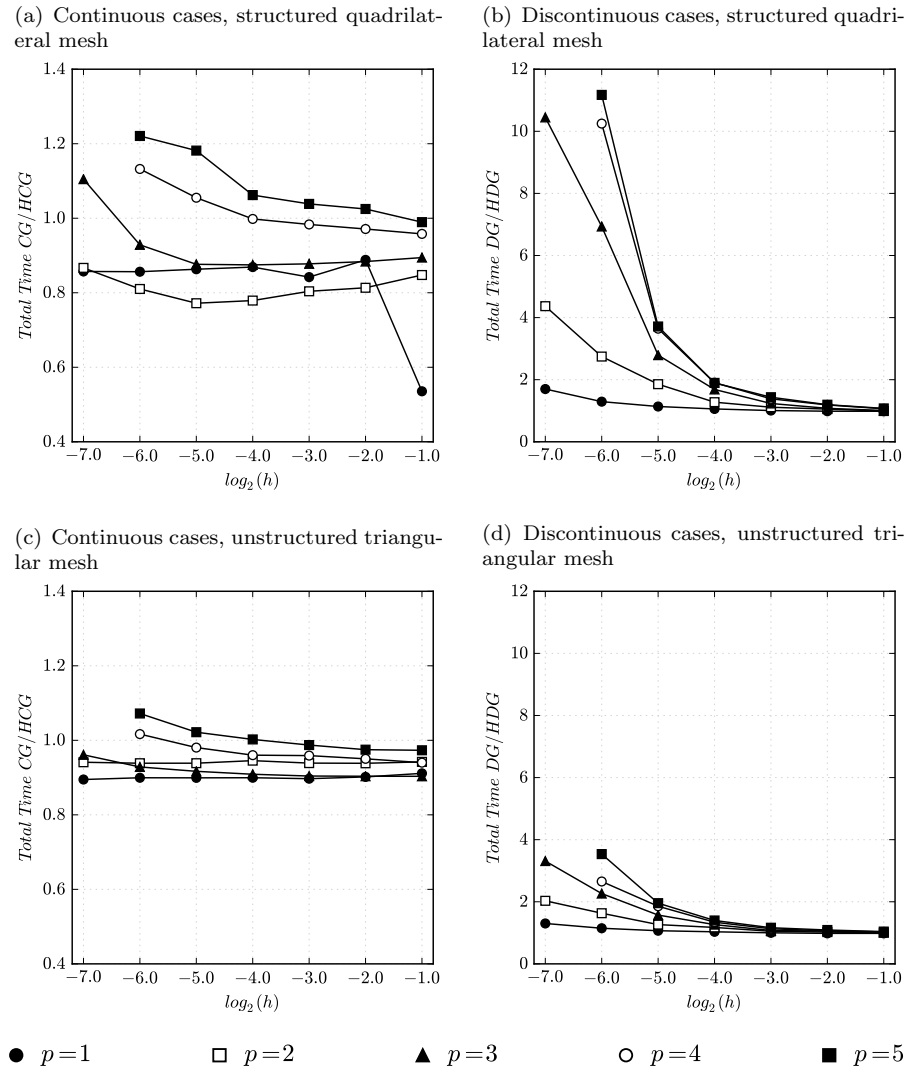


FIGURE 4.7: Impact of hybridization on the total time.

4.2.2 Time Consumption

We analyze the influence of hybridization on the time requirements. Figures 4.7 and 4.8 show the the ratio between the hybrid and non-hybrid solvers in terms of total and solving time consumptions respectively. Results depend on the type of considered mesh, on the polynomial degree p and on the mesh element size h. Let's recall that the total time accounts for the time for the initialization, the building of the linear system, the solving and, for discontinuous schemes, the post-process. The solving time refers to the time for solving the global system and, for hybrid schemes, also to the time for executing the local solver on each element. Each temporal data corresponds to the minimum CPU-time over four separate runs.

Regarding to the solving time from figure 4.8, we point out that the hybridization of the global system saves *solving* time in continuous schemes if $p \geq 3$, and in discontinuous

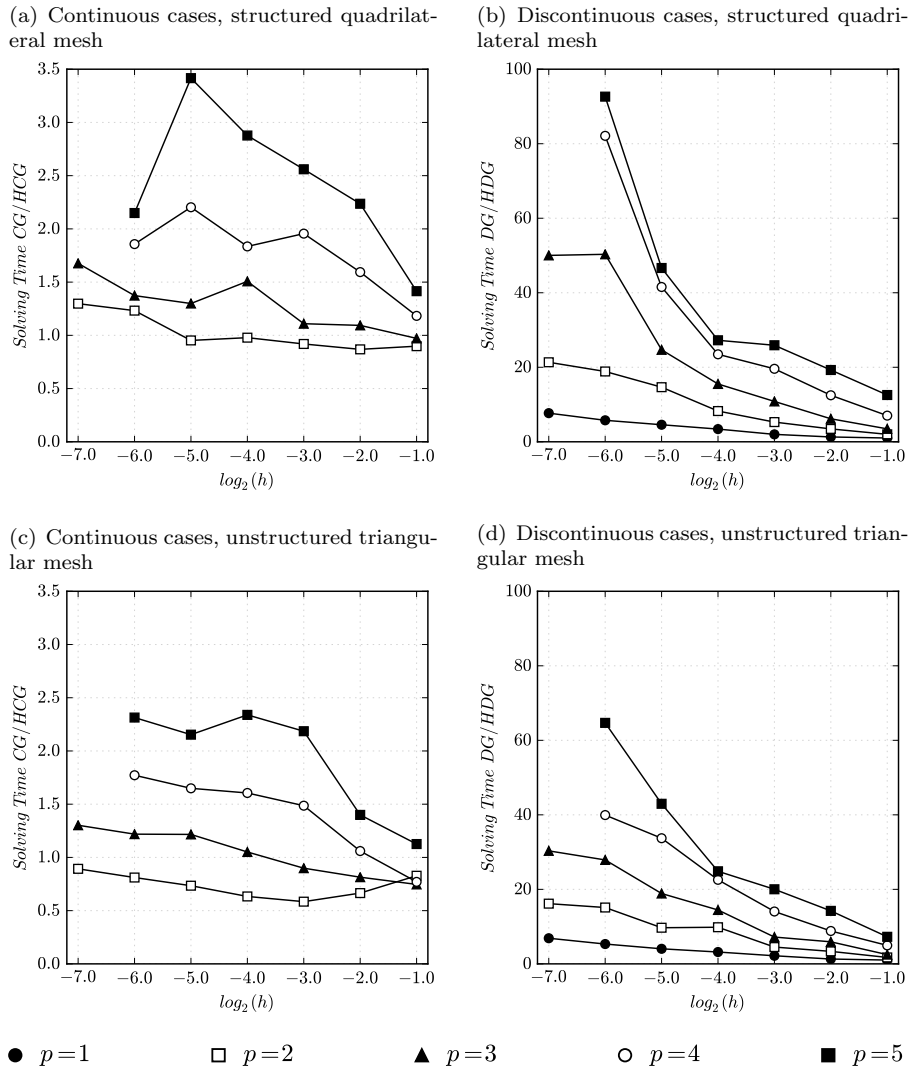


FIGURE 4.8: Impact of hybridization on the solving time.

schemes in any case. Hybridization leads to solving a smaller global system, but then each element is locally computed afterwards. The total solving time is overall reduced in discontinuous schemes since the size reduction of the global system is very remarkable. For high interpolation degrees in continuous schemes, or for any interpolation degree in discontinuous ones, the solving time is the part of the code that takes more time to run; therefore, the *total* amount of time required to get the solution is reduced in these cases.

In the continuous case, the difference in time is not that relevant. However, in the discontinuous case, the consideration of hybrid schemes reduces *dramatically* the solving and total times for fine high-order meshes. For instance, regarding figure 4.8, we find that for $p = 5$ simulations can become up to 11 times *faster* on quadrilateral meshes if hybrid schemes are considered; on top of that, the time saving grows exponentially with the mesh size h, meaning that high-order discontinuous schemes are not competitive if

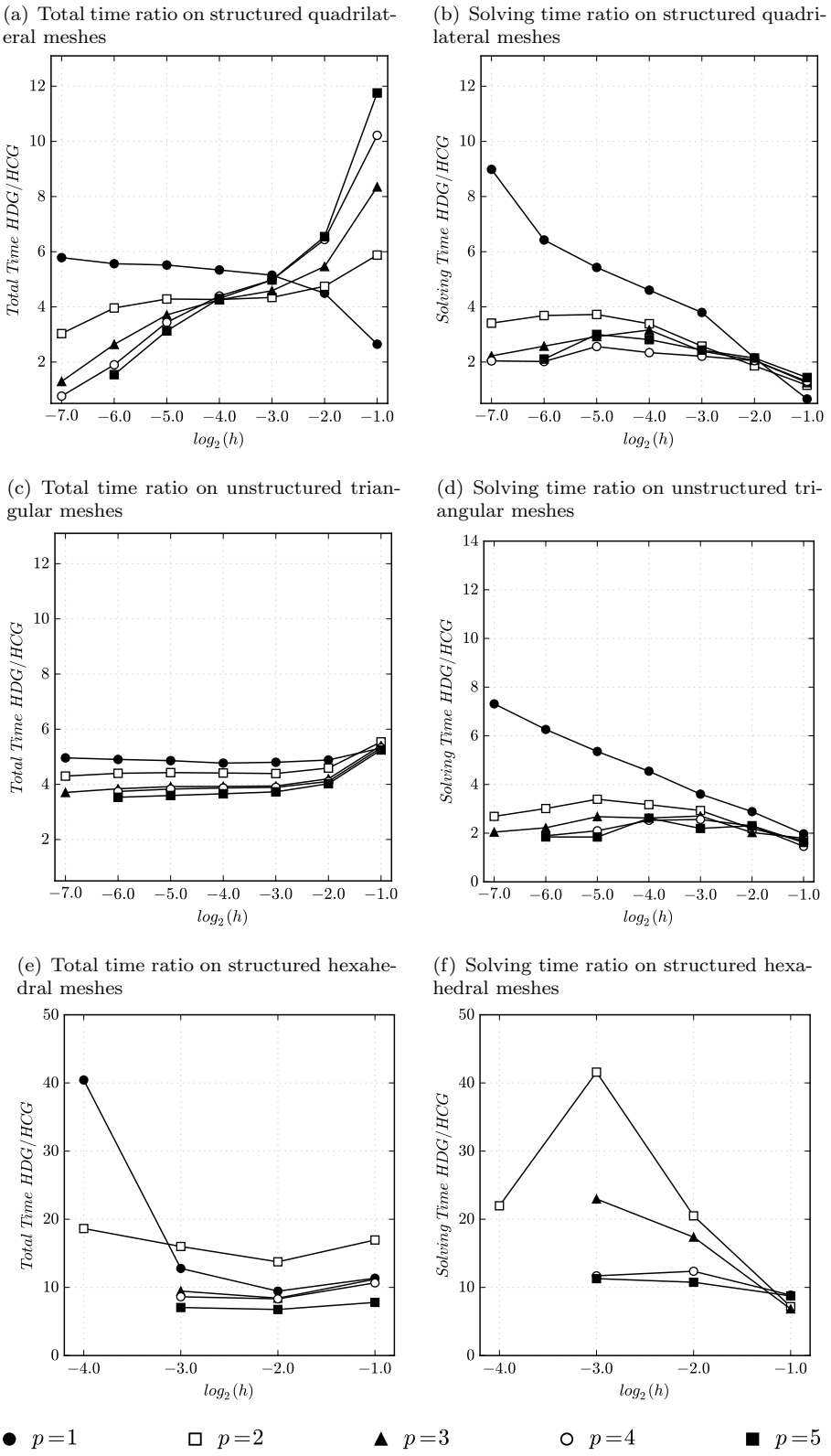


FIGURE 4.9: Comparison of total time (left column) and solving time (right column) consumptions between HCG and HDG solvers.

hybridization is not considered.

Secondly, we can compare the hybrid continuous and discontinuous approaches. Figure 4.9 presents HCG and HDG time ratios. Results show that in the proposed implementation the formers are faster than the latters, both in solving and total times. Discontinuous schemes take more time to build the system since they hold more unknowns than the continuous ones prior to the hybridization, and take more time to solve it since some nodes hold more than one unknown in the global system. Moreover, an additional time is spent getting the post-processed solution u_{post} within each element of the mesh. For fine high-order meshes, this difference gets decreased but is still remarkable. In these cases HDG is around 3-4 times slower than HCG for 2D, and around 7 times slower for 3D.

We highlight that time consumptions depend on the code implementation. Although the code is implemented in an efficient way, it can be further optimized as section 5.3 explains. Therefore, the time analyses presented here are just referring to our own implementation. Other authors could get different performances, but our conclusions would not differ significantly from those of other authors.

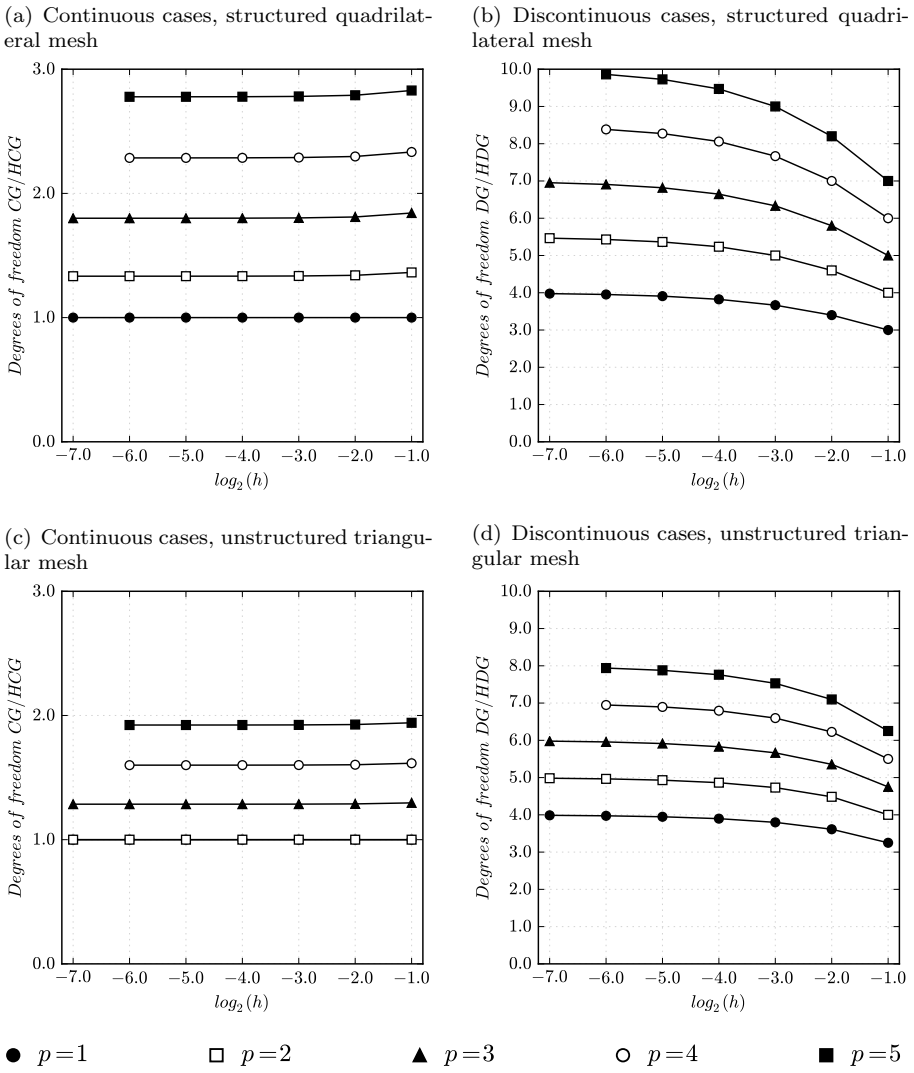


FIGURE 4.10: Impact of hybridization on the number of unknowns in the system.

4.2.3 Number of Unknowns

Figure 4.10 shows the ratio between hybrid and non-hybrid schemes in terms of the number of unknowns in the global system. Since the hybridization condensates the inner elemental nodes in the case of HCG, and the unknowns u and $\boldsymbol{q}$ in the case of HDG, the system gets reduced in any case. This reduction is specially noticeable in discontinuous schemes. For instance, if we consider a fine mesh with polynomial shape functions of degree 5, a reduction of a 90% is given in the discontinuous scheme if quadrilateral meshes are considered, and a 87% with triangular meshes. Considering the continuous scheme in the same situation, the number of unknowns gets a reduction of a 66% with quadrilateral meshes and a 50% with triangular meshes.

Figure 4.11 compares the number of unknowns in hybrid continuous and discontinuous schemes. We note that the rank in HDG schemes is always larger than in HCG. However,

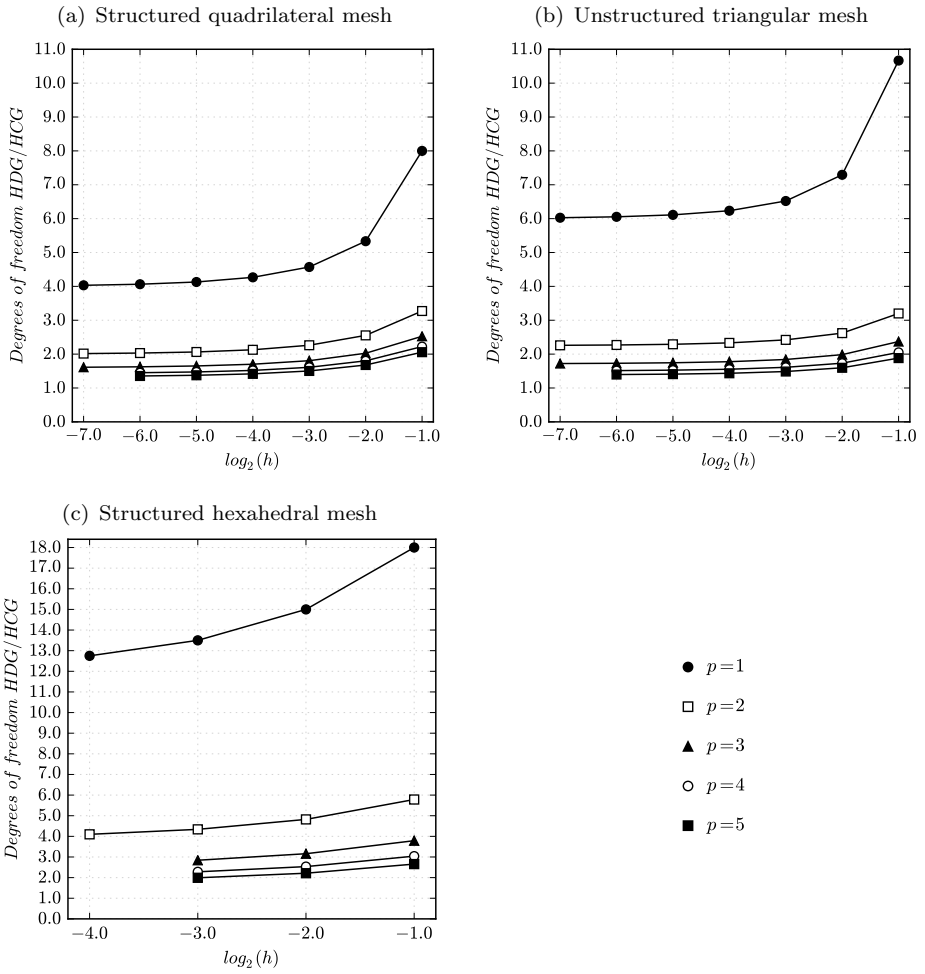


FIGURE 4.11: Comparison of number of unknowns between HCG and HDG solvers.

considering high-order meshes, both approaches get closer; for $p = 5$, 2D meshes lead to a factor of 1.5, while the hexahedral mesh leads to a factor of 2.

4.2.4 Sparsity

Figure 4.12 shows the ratio between hybrid and non-hybrid schemes in terms of the sparsity index, s_i. Let us recall that when s_i tends to 1, the matrix tends to be dense, and conversely when s_i tends to 0, the matrix tends to be sparse. We are interested in this second case, and this is given for high-order fine meshes.

As figure 4.12 shows, hybridization increases the sparsity index of the matrix. It leads to larger sparsity indexes s_i in all the studied cases, and this effect grows when p is increased. For instance, fine meshes with high interpolation degrees ($p = 5$) double the sparsity index on quadrilateral meshes and almost triple it on hexahedral meshes. However, this is a normal consequence of reducing the rank of the system. Overall,

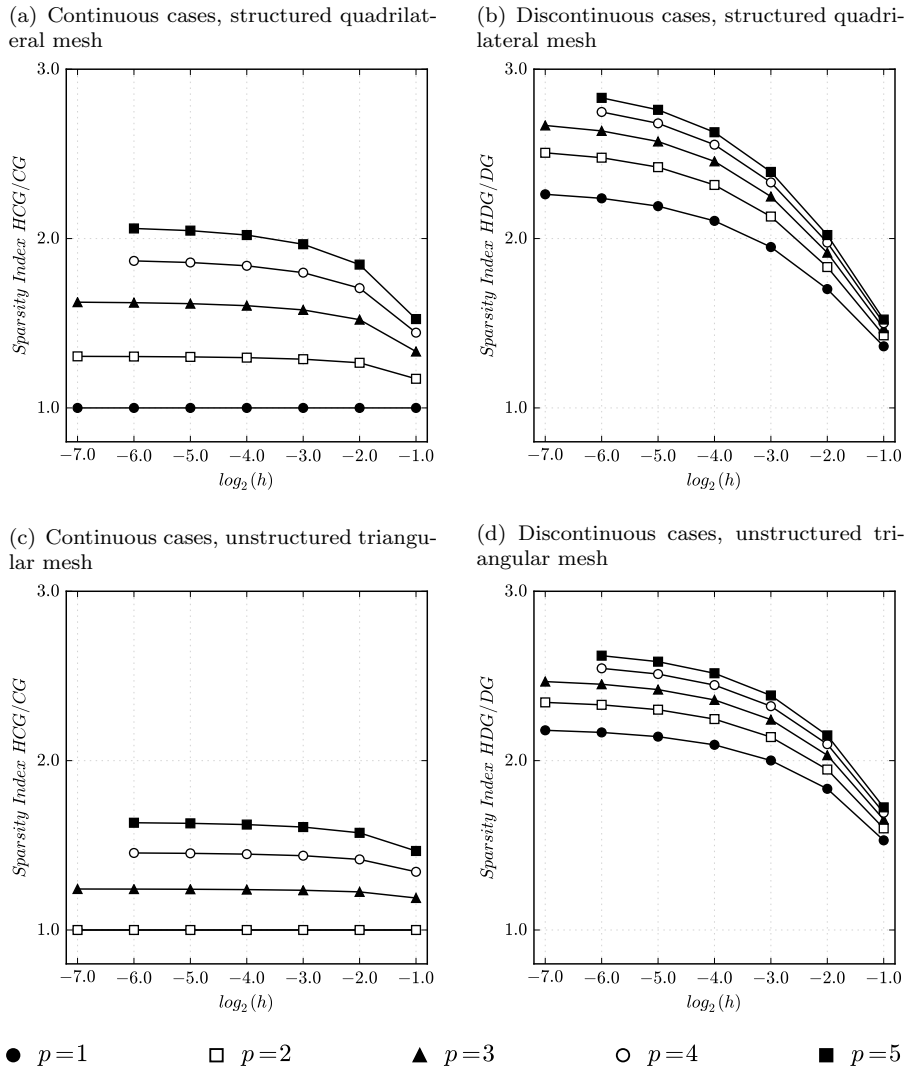


FIGURE 4.12: Impact of hybridization on the sparsity indices of the system.

the matrix entries are reduced in any case, so both memory and time are saved using hybridization.

We can compare hybrid continuous and discontinuous schemes in this terms (see figure 4.13). We note that the sparsity index in HDG schemes is always lower than in HCG ones, even when considering high order meshes. Figure 4.14 (directly extracted from [15]) shows the coupling of the unknowns in the HCG (denoted in fig. 4.14 as CG) and HDG formulations. On a discontinuous scheme, each node of a numerical trace is coupled with the nodes of the traces of the adjacent elements: on quadrilateral meshes, this means that each equation relates *seven* traces, and on triangular meshes *five* traces. Conversely, on a continuous scheme, the node on a vertex of an element is related with the nodes on the boundaries of all the adjacent elements: this means that some equations can relate up to *twelve* elemental boundaries for both quadrilateral and triangular meshes.

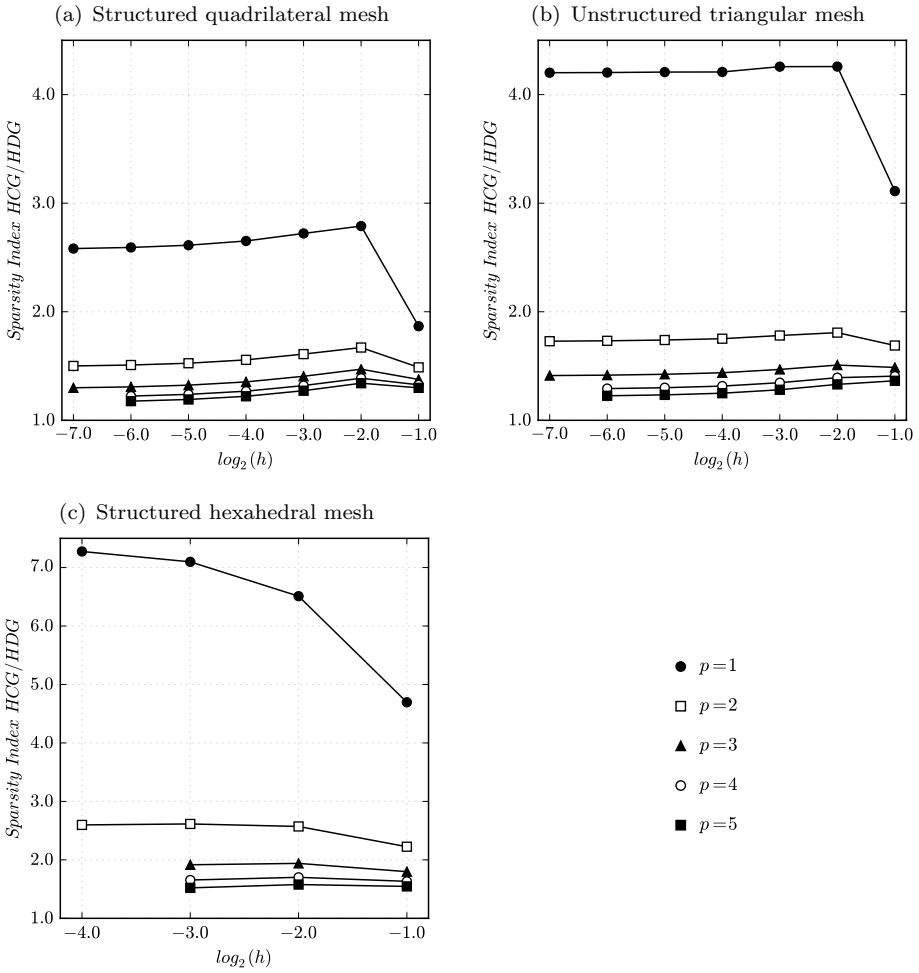


FIGURE 4.13: Comparison of sparsity indices between HCG and HDG solvers.

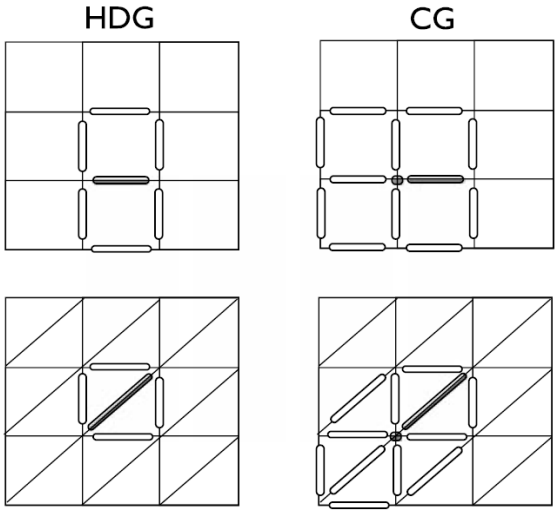


FIGURE 4.14: Coupling of any node within a trace in HDG schemes vs. coupling of a vertex node in HCG schemes, both in quadrilateral and triangular meshes. Figure extracted from [15].

4.3 Considering Real World Complex Geometries

This section considers complex geometries, such as round-shaped boundaries, inner holes and concave/convex shapes. The aim of this section is to also validate our software on this kind of domains.

In order to do so, two different examples are considered; the first one (see section 4.3.1) reproduces the real 2D-geometry of a brake disk. The second one (see section 4.3.2) considers a 3D-geometry of one half of a gear.

4.3.1 2D Example: Brake Disk

Our aim is to solve the elliptic diffusion problem with an homogeneous diffusion tensor ($\mathbf{D} := \mathbf{I}_2$). The boundary edges hold Dirichlet boundary conditions, which have been properly selected together with the source term f in order to give an exact solution of the form

$$u(x, y) = \sin(2\pi k x) \cdot \sin(2\pi k y),$$

with $k = 1/100$.

We have considered a curved high-order mesh with an interpolation degree of $p = 7$, see figure 4.15. This mesh is able to describe curved shapes with not many elements achieving a high level of accuracy.

Figure 4.16(a) shows the analytical solution.

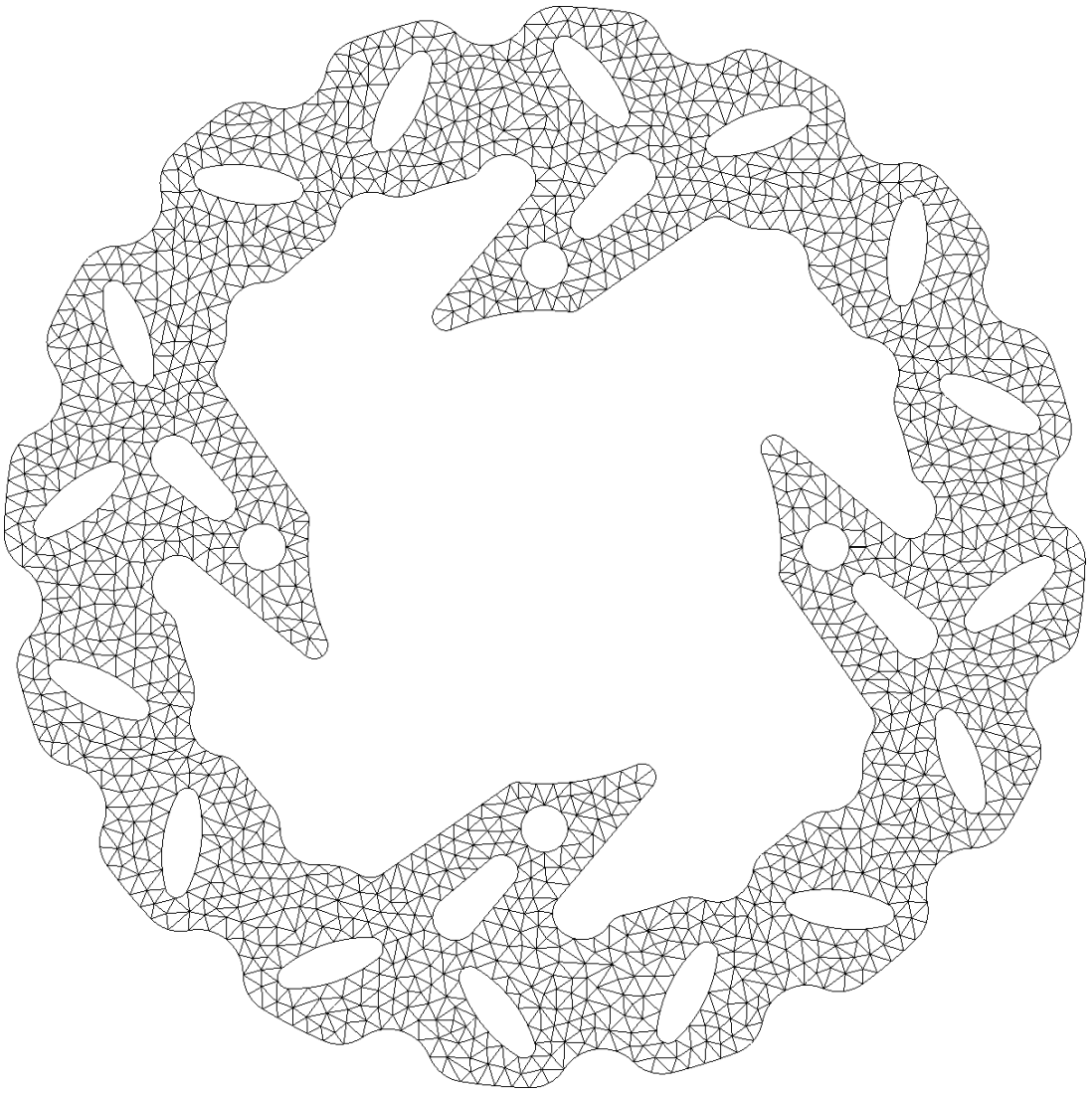
The error of this solution depends on the solver considered. Table 4.3 presents the L_2-error for HCG and HDG (with $\tau = 1$) formulations. Note that continuous and discontinuous formulations achieve the same order of accuracy for the scalar unknown u. However, the flux unknown $\boldsymbol{q}$ has better accuracy in HDG, since the convergence rate is one order above the one of HCG. Moreover, the L_2-error is highly reduced if local post-process technique is considered.

	u	u_{post}	$\boldsymbol{q}$
HCG	$3.02 \cdot 10^{-4}$	-	$2.00 \cdot 10^{-3}$
HDG	$1.95 \cdot 10^{-4}$	$3.98 \cdot 10^{-5}$	$2.24 \cdot 10^{-4}$

TABLE 4.3: L_2-error of the brake disk, depending on the solver.

The error is not homogeneously distributed within the domain; figures 4.16(b) to 4.16(d) show the spatial distribution of the error, depending on the solver. Figure 4.17 shows the errors in a detailed zone of the geometry for both solvers.

(a) Full view of the mesh



(b) Detail of the mesh

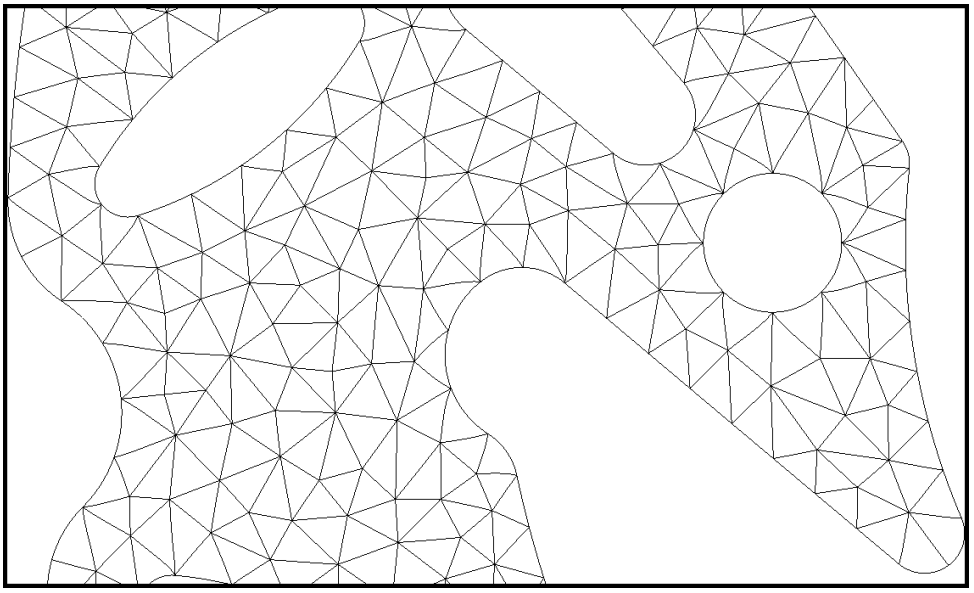
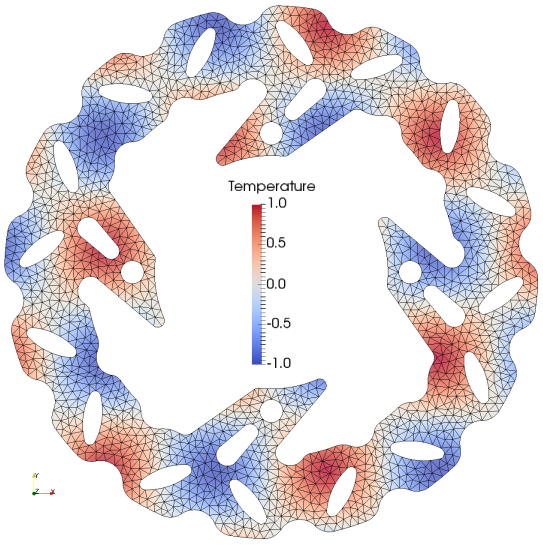
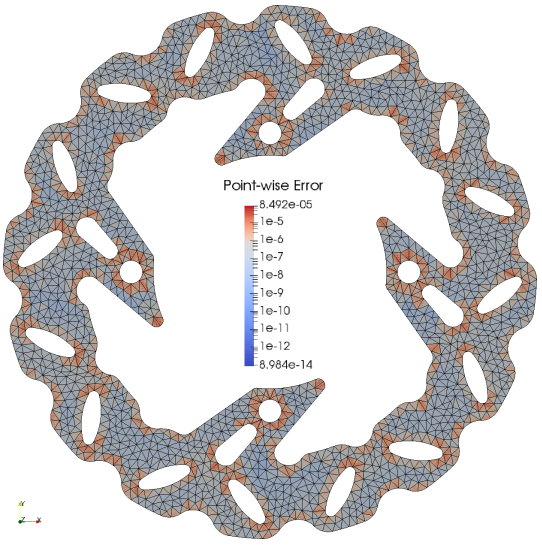


FIGURE 4.15: 2D triangular mesh of the brake disk.

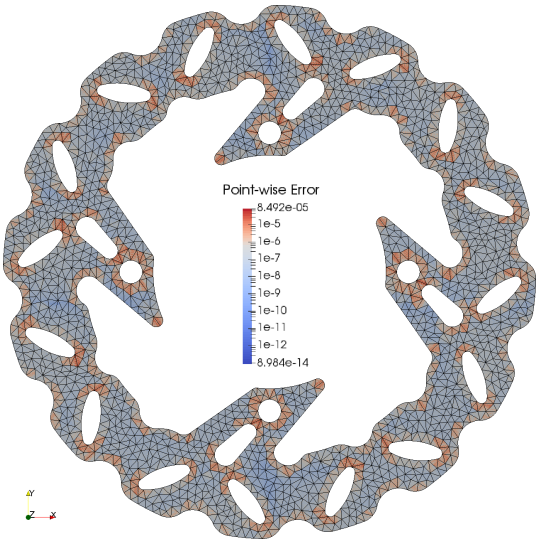
(a) Analytical solution to the brake disk



(b) Point-wise error of the brake disk with the HCG solver



(c) Point-wise error of the brake disk with the HDG solver



(d) Point-wise error of the brake disk with the HDG solver, post-processed solution

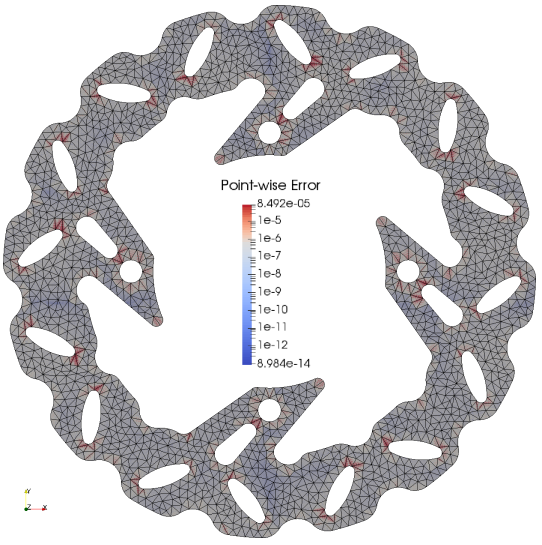
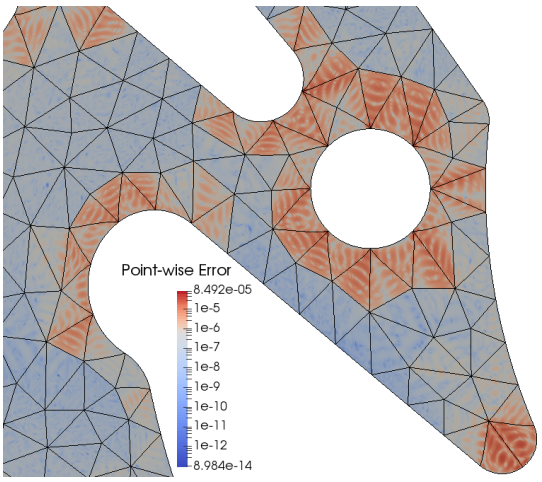


FIGURE 4.16: Brake disk: scalar solution and its point-wise error spatial distribution.

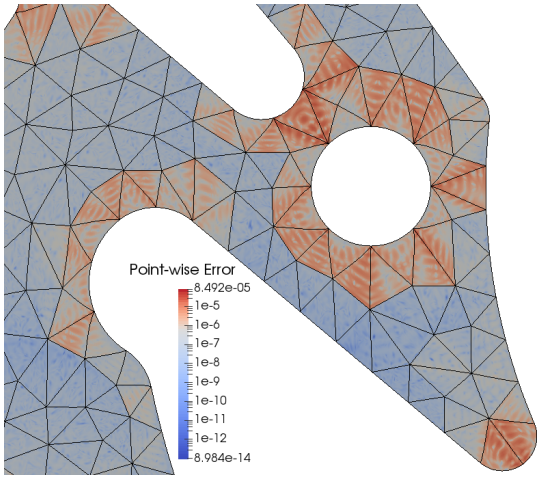
Note that, in all cases, point-wise errors are mostly concentrated within the elements around the curved boundaries of the mesh, since the spatial discretization errors arise mainly there.

The maximum value of the error within all the domain is below $8.5 \cdot 10^{-5}$ in any case, which means that the solvers can handle curved boundaries, despite the additional errors that arise due to the consideration of curved shapes. Figure 4.16(d) shows the point-wise error of the post-processed scalar solution, and figure 4.17(c) details the zoom including a zone with curved-shape boundaries. Comparing the post-processed point-wise errors with the non post-processed ones, we note that local post-processing reduces the point-wise error within all the domain, but specially around the curved boundaries.

(a) HCG solver



(b) HDG solver



(c) HDG solver, post-processed solution

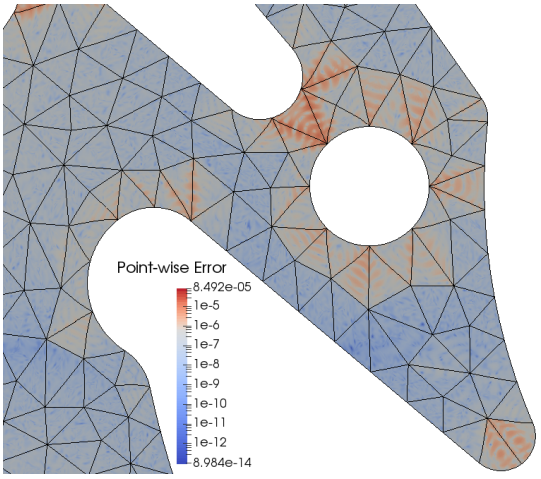


FIGURE 4.17: Details of the point-wise error of the brake disk, depending on the solver.

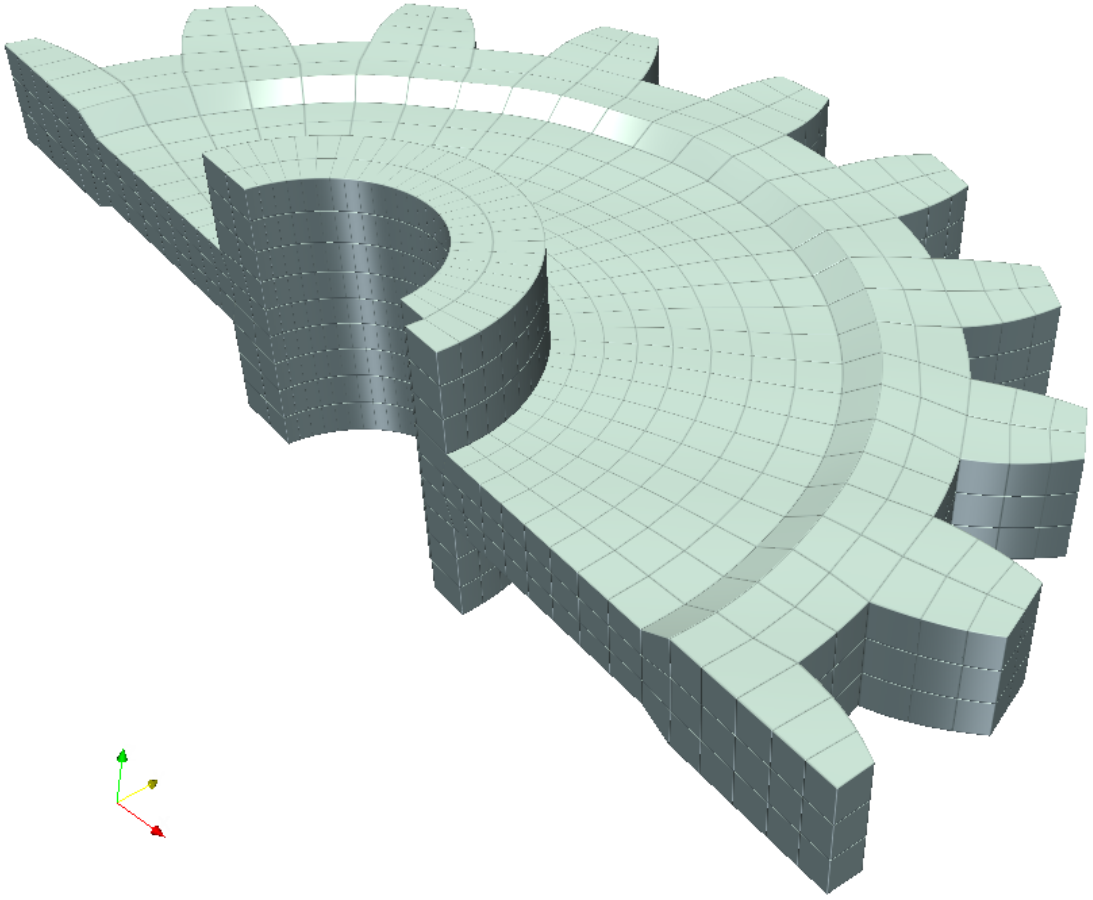
Let us recall that the local post-processing does *not* consider a different interpolation for the geometry, but only for the solution. Therefore, the spatial discretization errors still remain. The total error is highly reduced, since the solution is interpolated with polynomials of an extra degree $(p+1)$.

4.3.2 3D Example: Semi Gear

This section considers the 3D-geometry of a half of a gear (see figure 4.18(a)).

Our aim is to solve the elliptic diffusion problem with an homogeneous diffusion tensor ($\mathbf{D} := \mathbf{I}_2$) considering symmetry Neumann boundary conditions in the faces at $y = 0$ and Dirichlet boundary conditions along the rest of the faces. They have been properly

(a) Full view of the mesh



(b) Detail of the mesh

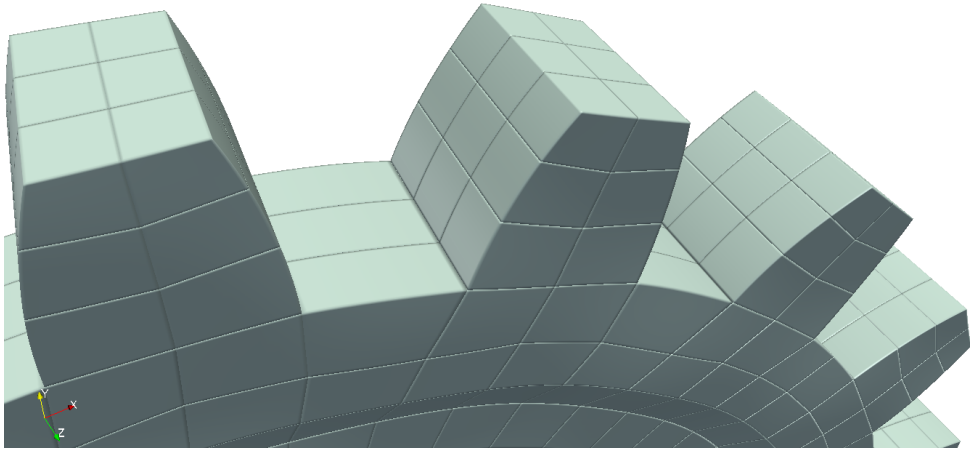


FIGURE 4.18: 3D hexahedral mesh of one half of a gear.

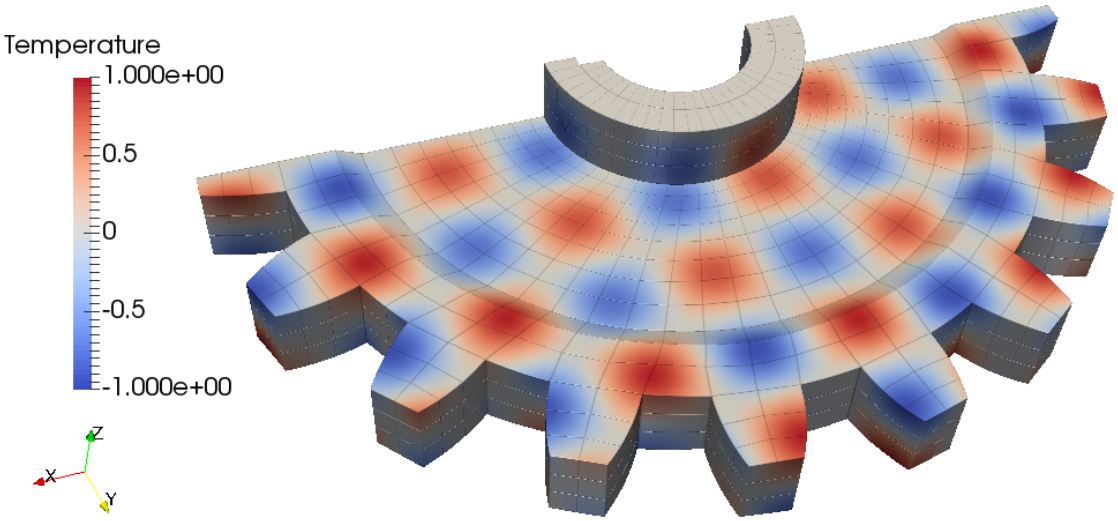


FIGURE 4.19: Analytical solution to the semi gear.

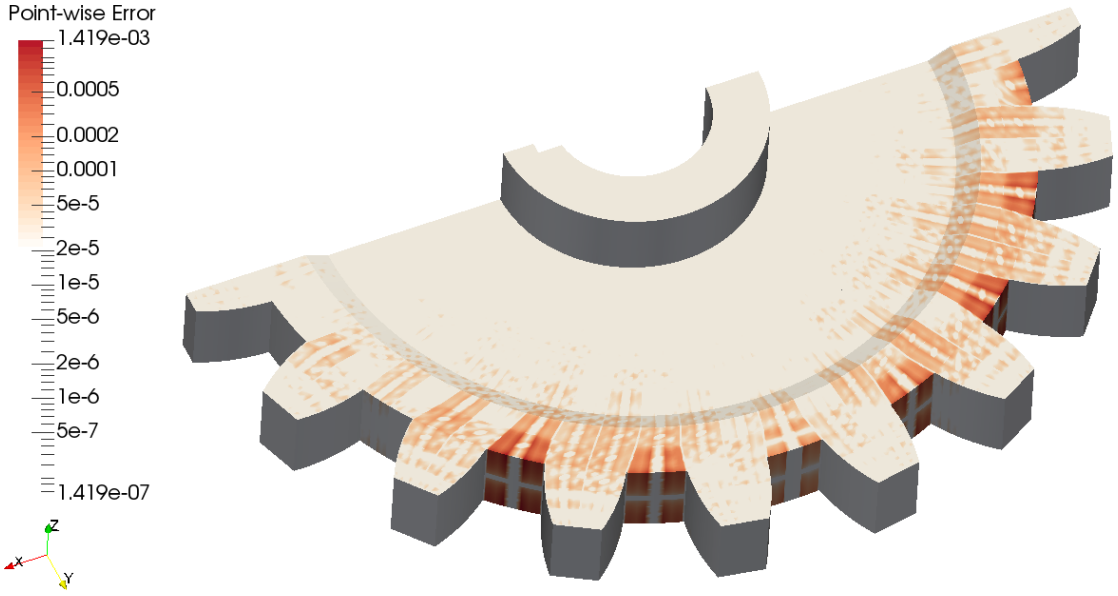


FIGURE 4.20: Point-wise error of the semi gear with the HCG solver.

selected together with the source term f in order to give an exact solution of the form

$$u(x, y, z) = \sin(2\pi kx) \cdot \sin(2\pi ky) \cdot \sin(2\pi kz),$$

with $k = 1/25$.

We solve this problem using a coarse curved high-order mesh of interpolation degree $p = 4$, see figure 4.18. This mesh can deal with curved boundaries using few elements (see detail of the mesh in figure 4.18(b)).

Figure 4.19 shows the analytical solution of the problem.

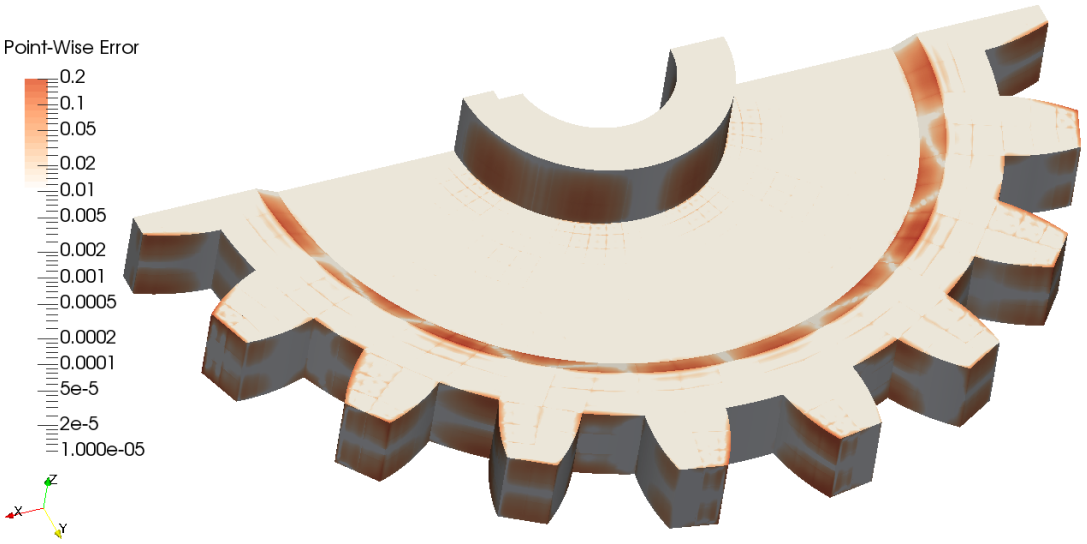


FIGURE 4.21: Point-wise error of the semi gear with the HDG solver.

Table 4.4 presents the total L_2-error of the HCG and HDG (with τ=1) solvers. In this case, the continuous solution achieves higher accuracy than the discontinuous one for both the scalar unknown u and the flux unknown $\boldsymbol{q}$.

	u	u_{post}	$\boldsymbol{q}$
HCG	$1.38 \cdot 10^{-2}$	-	$2.45 \cdot 10^{-2}$
HDG	$1.00 \cdot 10^{+1}$	$8.51 \cdot 10^{+0}$	$1.10 \cdot 10^{+1}$

TABLE 4.4: L_2-error of the semi gear, depending on the solver.

The point-wise error of the scalar solution is shown in figure 4.20 for the HCG case, and in figure 4.21 for the HDG case. These figures show that the point-wise error is also not homogeneously distributed within all the domain; it is larger on curved faces for the same reason explained in the 2D brake disk example (see section 4.3.1). However, the point-wise error distribution depends on the solver. Note that the error with the HDG solver is relatively high at faces with confronted curved edges, even where Dirichlet boundary conditions have been prescribed. A way to reduce the local error on this kind of elements would be dividing these elements in two, leading to a straight shared face. This procedure would facilitate the interpolation of the solution within these elements. In any case, both solvers tend to provide the correct solution.

4.4 Considering non-Homogeneous Domains

In this section we focus on non-homogeneous domains, in order to show that the solver also works for any symmetric positive definite diffusion tensor $\mathbf{D}$. Two kind of domains are considered: anisotropic and heterogeneous domains.

Section 4.4.1 considers anisotropic domains, meaning that the diffusion tensor achieves different values in the principal directions. Section 4.4.2 considers heterogeneous domains, meaning that the diffusion tensor *depends* on the spatial coordinates.

From now on, we will refer to the hydraulic diffusion problem in order to illustrate the simulations.

Both sections consider the same geometry, a square $[0, 1] \times [0, 1]$, with null source term ($f = 0$):

- The edge at $x = 0$ is modeled as an impermeable layer, so it holds a Neumann boundary condition set to zero ($g_N = 0$ at $x = 0$).
- The edge at $x = 1$ is modeled as a drain, so it holds a Neumann boundary condition set to one ($g_N = 1$ at $x = 1$); in this way, some water is allowed to escape the domain.
- The edges at $y = 0$ and $y = 1$ represent two different water reservoirs with constant water levels, which hold Dirichlet boundary conditions, set to 0 and 1 respectively ($g_D = 0$ at $y = 0$; $g_D = 1$ at $y = 1$).

The hydraulic head at the top reservoir is maximum. Therefore, water is forced to flow across the domain, until it reaches the bottom reservoir (with lower head) or until it escapes the domain through the drain in the right boundary.

The diffusion tensor accounts now for the permeability of the soil; therefore, the water will flow across the domain according to $\mathbf{D}$.

In order to solve this problem, the high-order mesh with an element size of $h = 1/64$ and a polynomial degree $p = 6$ has been considered, which is accurate enough for our purposes.

4.4.1 Anisotropic domain

In this section we consider an anisotropic permeability tensor $\mathbf{D}$. This kind of permeability tensors can always be expressed in terms of the values (D_1, D_2) along the principal directions in the following way:

$$\mathbf{D}' = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}, \tag{4.2}$$

if the principal directions of the tensor are aligned with the x- and y-axis. In order to get the permeability tensor *oriented* in a given direction, we have to *rotate* the tensor using:

$$\mathbf{D} = \mathbf{R}^T \cdot \mathbf{D}' \cdot \mathbf{R}, \tag{4.3}$$

where $\mathbf{R}$ is the *2D rotation matrix*, which depends on the (counterclockwise) angle of rotation α:

$$\mathbf{R} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}. \tag{4.4}$$

In this way, we are able to define any constant permeability tensor. This way of constructing $\mathbf{D}$ leads to a symmetric matrix; note then that both D_1 and D_2 have to be positive ($D_1, D_2 > 0$) in order to get a symmetric *positive definite* tensor. Picking $D_1 = D_2$ we end up with the isotropic case; the anisotropy arises when $D_2 \neq D_1$.

The next example considers the domain explained in 4.4 with diagonal stratification of the soil, from northwest to southeast ($\alpha = \frac{\pi}{2}$).

Different (D_1, D_2) tuples have been considered:

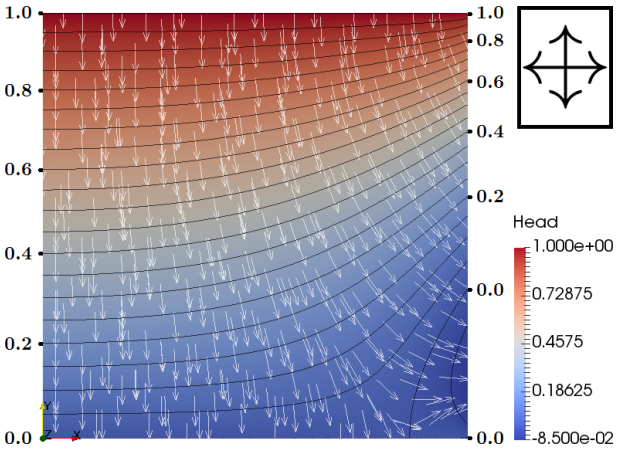
- Isotropy: (D_1, D_2)=(1,1).
- Anisotropy: (D_1, D_2)=(1,10).
- Strong anisotropy: (D_1, D_2)=(1,100).

We proceed now to simulate the hydraulic diffusion problem on these 3 scenarios.

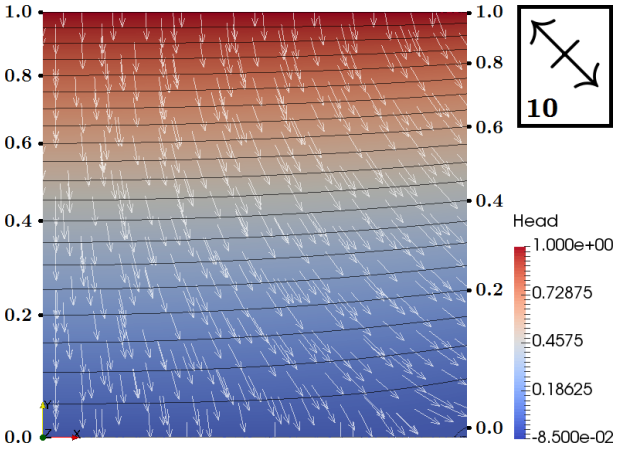
For the three cases, figure 4.22 shows the resulting hydraulic head, represented by the contour fill and the contour lines, and the flow direction is denoted with white arrows. We highlight that the white arrows account for the flow *direction* only, and not for its magnitude; instead, the flow magnitude is shown in figure 4.23, together with the stream lines of the water flow.

In the isotropic case, some water coming from the top reservoir flows towards the bottom reservoir, and some other amount of water ends up flowing towards the drain on the right boundary. Note that the drain is even able to catch a small amount of water from the bottom reservoir too, creating an inversed flow upwards at the right bottom corner. The equipotential lines defined by the hydraulic head in figure 4.22 and the stream lines defined by the flow in figure 4.23 are orthogonal *just* in this isotropic case, as expected. As the stratification of the soil gets increased, the equipotential lines tend to become parallel and horizontal, and the water in the bottom reservoir losses capacity to flow towards the drain. Regarding to the flow magnitude in figure 4.23, we observe that it

(a) Head in isotropy



(b) Head in a 1:10 anisotropy



(c) Head in a 1:100 anisotropy

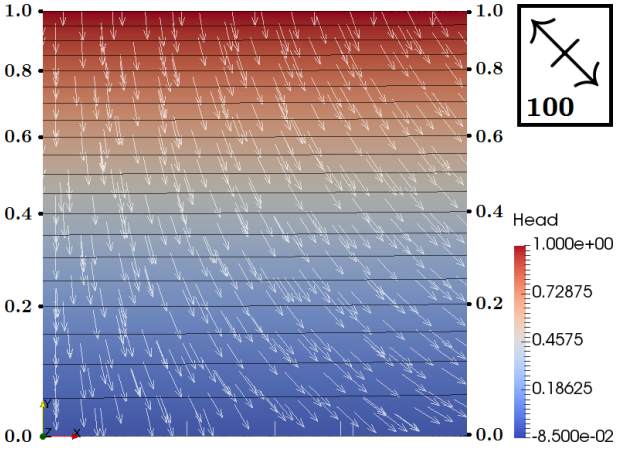
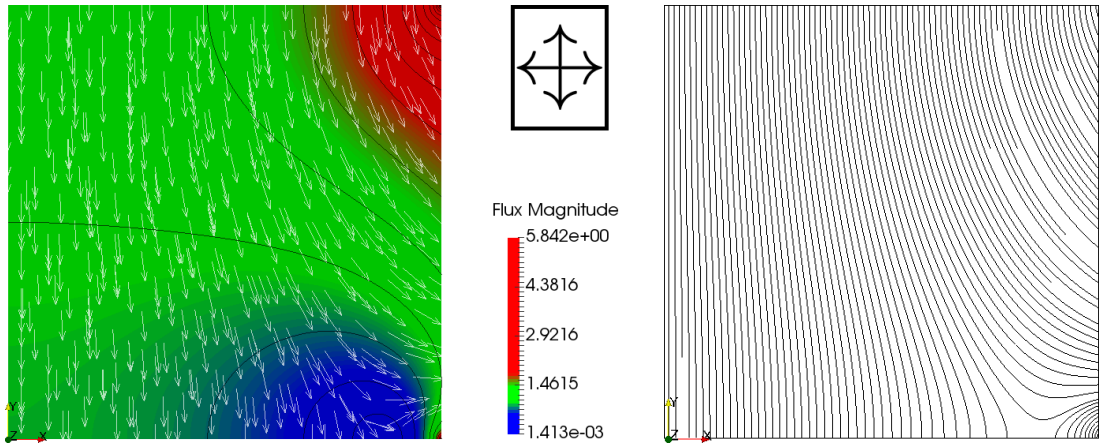


FIGURE 4.22: Head (contour fill and contour lines) and flow direction (white arrows) in an anisotropic soil depending on the stratification degree.

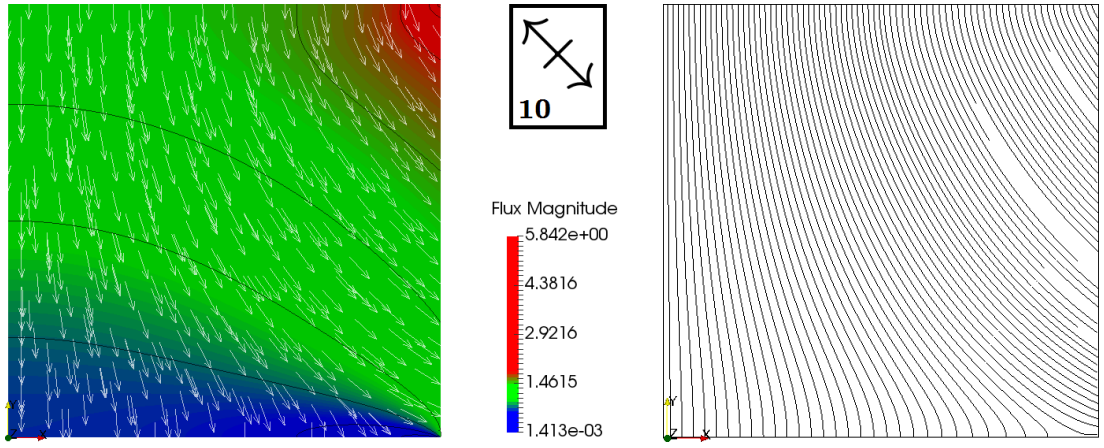
(a) Flow magnitude in isotropy

(b) Stream lines in isotropy



(c) Flow magnitude in a 1:10 anisotropy

(d) Stream lines in a 1:10 anisotropy



(e) Flow magnitude in a 1:100 anisotropy

(f) Stream lines in a :100 anisotropy

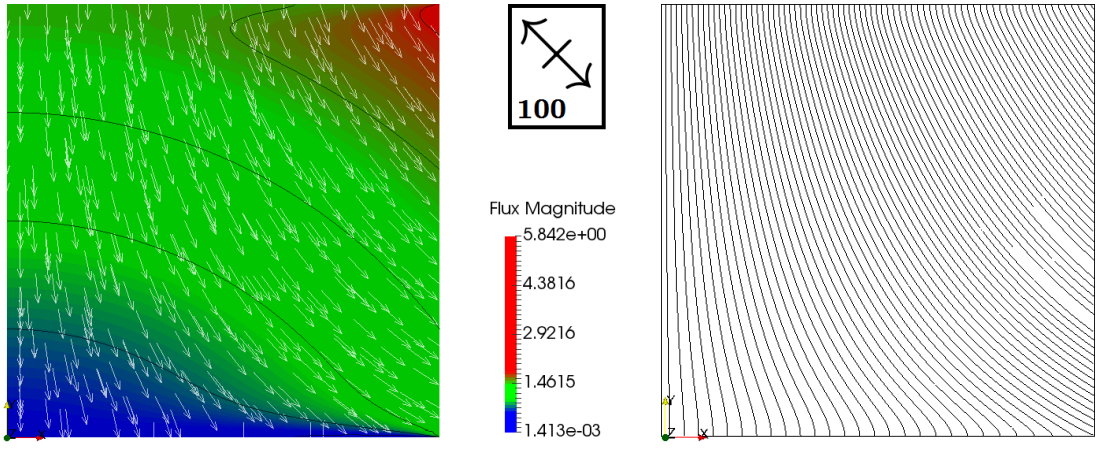


FIGURE 4.23: Flow magnitude (left column) and stream lines (right column) in an anisotropic soil depending on the stratification degree.

is quite uniform, presenting no oscillations. Just two areas (the right top corner and the front of the bottom reservoir) present higher and lower flows respectively, the first one due to the proximity of the reservoir to the drain, and the second one due to the influence of the impermeable layer at the left boundary, that forces the water to flow vertically even if it is not its preferential direction.

As a conclusion, we can say that the three scenarios provide different results depending on the permeability considered, which match with our previous expectations.

4.4.2 Heterogeneous domain

This last section considers an heterogeneous permeability tensor $\mathbf{D}$, which *does* depend on the spatial coordinate $\boldsymbol{x}$ and on the considered direction. In order to get this kind of diffusion, a possibility is to express $\mathbf{D}$ in terms of equations (4.2) and (4.3) defining coordinate-dependent D_1 and D_2 permeabilities.

Here, we consider the following ones:

$$D_1(x, y) = 1 + k_1 \cos^2(\omega\pi x), \tag{4.5a}$$

$$D_2(x, y) = 1 + k_2 \sin^2(\omega\pi y). \tag{4.5b}$$

In this way, we assure the positive definiteness of the tensor ($\{D_1, D_2\} \geq 1$) and its boundedness ($1 \leq D_1 \leq 1 + k_1$, $1 \leq D_2 \leq 1 + k_2$).

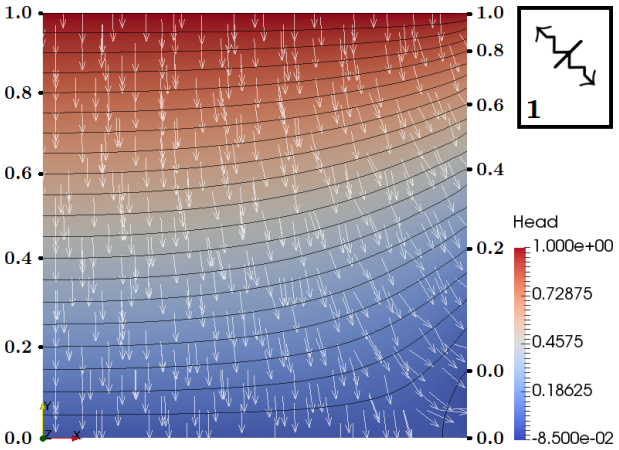
We have considered the same diagonal stratification of the soil as before, from northwest to southeast ($\alpha = \frac{\pi}{2}$). The oscillation frequency ω has been set to 8.44 to produce a high heterogeneity (8.44 oscillations of the permeability per unit of length in each diagonal direction).

Finally, we consider several values for the (k_1, k_2) tuples, which define the stratification degree within the heterogeneity of the soil:

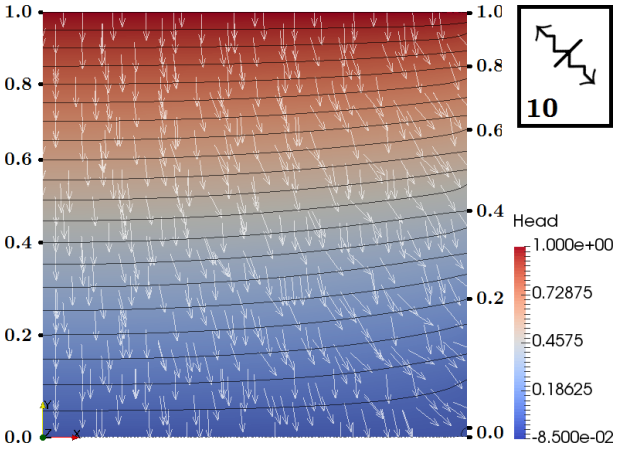
- Low stratification: (k_1, k_2)=(1,1). This leads to $1 \leq \{D_1, D_2\} \leq 2$.
- Moderate stratification: (k_1, k_2)=(1,10). This leads to $1 \leq D_1 \leq 2$ and $1 \leq D_2 \leq 11$.
- High stratification: (k_1, k_2)=(1,100). This leads to $1 \leq D_1 \leq 2$ and $1 \leq D_2 \leq 101$.

These 3 scenarios have been simulated; the resulting hydraulic heads are represented in figure 4.24 and the flow magnitude and the stream lines are shown in figure 4.25, in the same way as in section 4.4.1.

(a) Head in a 1:1 heterogeneity



(b) Head in a 1:10 heterogeneity



(c) Head in a 1:100 heterogeneity

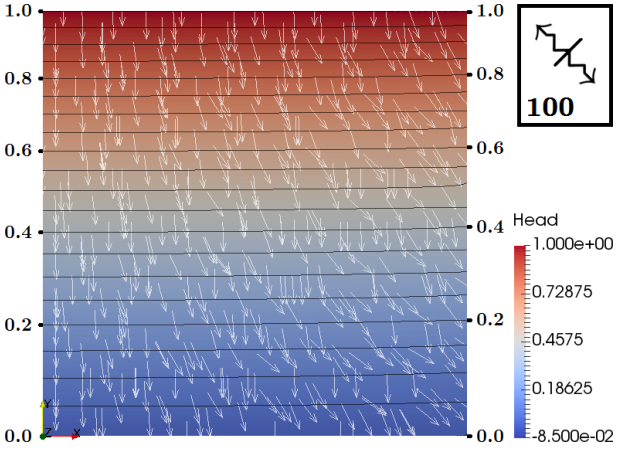


FIGURE 4.24: Head (contour fill and contour lines) and flow direction (white arrows) in an heterogeneous soil depending on the stratification degree.

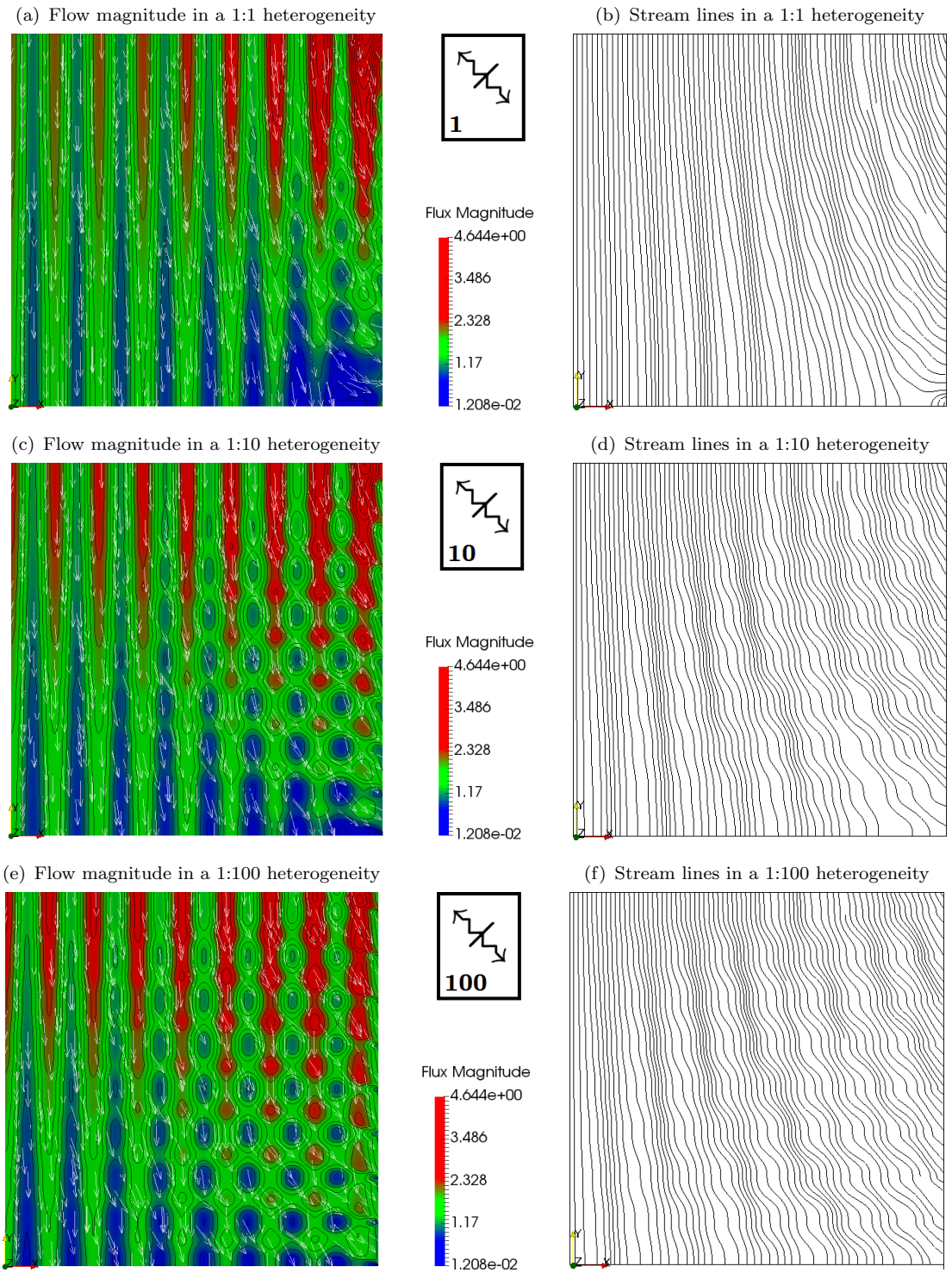


FIGURE 4.25: Flow magnitude (left column) and stream lines (right column) in an heterogeneous soil depending on the stratification degree.

The results of the simulations in this kind of heterogeneous domains are significantly different from those corresponding to the anisotropic cases presented in section 4.4.1, specially the water flow. The streamlines and the flow magnitude in figure 4.25 present an oscillatory behavior, that shows that the water is permanently changing its direction and its velocity while flowing downwards, according to the permeability of the soil. The oscillations decrease at the left part of the domain, due to the influence of the impermeable boundary at $x = 0$. As the permeability gets more oscillatory, the streamlines and the flow magnitude do so. The hydraulic head is shown in figure 4.24. It presents a similar behavior as in the anisotropic cases, with a tendency to get horizontal as the stratification increases. Note that the flux is not orthogonal to the equipotential lines in any of these heterogeneous cases.

The dependency of the results with respect to the permeability tensor considered in each case is then clearly highlighted in these examples.

As it has been shown with the anisotropic and heterogeneous scenarios, the finite element codes are able to deal with any symmetric positive-definite diffusivity.

Chapter 5

Conclusions and Future Work

5.1 Summary

We developed four efficient high-order finite element codes (CG, HCG, DG and HDG), able to solve stationary diffusion problems in 2D and 3D. These codes are based on the continuous and discontinuous Galerkin formulations, which can be hybridized by means of the static condensation procedure. They have been finally validated and analyzed to perform a comparison between them.

The chapters in this thesis answer the following questions:

Chapter 2:

- What is a stationary diffusion problem?
- How is the continuous Galerkin method formulation?
- How is the discontinuous Galerkin method formulation?
- What does it mean to hybridize a linear system?

Chapter 3:

- How are high-order meshes generated?
- How is the finite element code implemented?
- How are the results visualized?

Chapter 4:

- Is the code well implemented?
- Can it handle 2D and 3D simulations?

- Can it handle complex geometries?
- Can it handle non-homogeneous diffusion?
- How is the performance of high-order methods compared to linear ones?
- How is the performance of hybridized approaches compared to non-hybridized ones?
- How is the performance of the discontinuous approach compared to the continuous one?

Overall, we can say that we are very pleased with the achieved results and that we fulfilled all the objectives stated at the beginning of the thesis.

5.2 Conclusions

We have shown the features of continuous and discontinuous approaches, the effect of hybridization and the use of high-order meshing in triangular, quadratic and cubic meshes.

One the one hand, we have checked that *hybridization* optimizes the code in *all* the performance aspects. For continuous approaches, it makes sense just if high-order meshing is considered ($p > 1$), getting more interesting as p increases. For discontinuous approaches it is worthy with *any* interpolation degree, even for linear interpolations, since the scalar (u) and flux ($\boldsymbol{q}$) unknowns within the elements are then condensed. For both continuous and discontinuous approaches, but specially for this latter, hybridization saves memory and time and reduces the number of degrees of freedom of the global system. On top of that, we point out that hybridization does not affect the L_2-error of the solution. Therefore, our recommendation is to always use hybrid formulations.

On the other hand, we discuss about the use of *high-order unstructured methods*; section 4.1.2 highlights that, considering meshes with the same spatial resolution, high-order meshes lead to less L_2-error that linear meshes. In addition, the number of unknowns in the system decreases for continuous and discontinuous cases when high-order meshes are considered, with the exception of the non-hybridized continuous case, for which it makes no difference to use high- or low-order polynomial degree. With this information in hand, we note that high-order methods are more interesting than linear methods, and thus it leads to recommend the use of high-order discretization. Moreover, high-order meshes are able to catch curved-shape geometries better than linear meshes, requiring no local mesh refinement.

Finally, we compare *continuous* and *discontinuous schemes*, considering their hybridized high-order versions. The main conclusion is that, for the elliptic problem, HCG and HDG are both competitive. The HDG code turns out to be less efficient than the HCG one in terms of memory and time consumptions; moreover, it leads to a larger rank of the global system to be solved and also leads to less sparse global matrices. However, the HDG method obtains a flux unknown $\boldsymbol{q}$ with an extra order of convergence $(p+1)$ compared to the flux in HCG, which is not directly computed, just derived from the scalar unknown u, converging with an order of p. On top of that, local post-processing of the scalar unknown u leads to an extra order of convergence $(p+2)$ for the scalar unknown too. Thus, we can say that HDG requires higher computational resources than HCG, but conversely it provides more accurate solutions of both the scalar unknown and its flux.

5.3 Future Work

The current implementation of the four methods provides accurate numerical results. However, they can still be further optimized in order to improve its performance in terms of time and memory consumptions.

On the one hand, the *elemental contributions* in hybridized schemes (HCG and HDG) could be constructed directly, instead of applying the static condensation technique to the non-hybridized linear systems; these procedures are explained in [14] for HCG and in [15] for HDG. These ways of constructing hybridized local contributions require a deep knowledge of the block-structure of the elemental contributions, so its implementation is somewhat more complex than ours. Since they make use of additional auxiliary arrays, they may require more memory, but they may save computational time during the building of the global system instead.

On the other hand, the *nodes* could be *renumbered* prior to solving the system in the four finite element codes. This technique would lead to a lower band-width which reduces the memory of the ILU preconditioner, as explained in [19]. Moreover, sparse direct solvers could be considered instead of iterative ones, since the memory requirements would decrease.

Parallel computation will be considered in the near future. This will reduce the computational cost of the methods. The parallel computation in hybrid schemes is relatively easy to implement for the local solvers, since the unknowns within each element do not depend on the ones of the other elements. Local post-processing of mixed schemes can as well be easy parallelized for the same reason.

Finally, I want to recall that my mentors will extend all this work to convective formulations in the near future.

Appendix A: Meshes used in the computations

This appendix contains the meshes used in section 4.1.

Figure A.1 shows the structured quadrilateral meshes for the 2D squared domain in section 4.1.1. These meshes coincide as well with the faces of the elements in the 3D cubic mesh that are contained in the plane $Z = 0$.

Figure A.2 shows the unstructured triangular meshes for the 2D squared domain.

Figure A.3 shows the structured quadrilateral meshes for the 2D squared domain in section 4.1.2. These meshes have all the same h/p ratio, and thus they have the same spatial resolution.

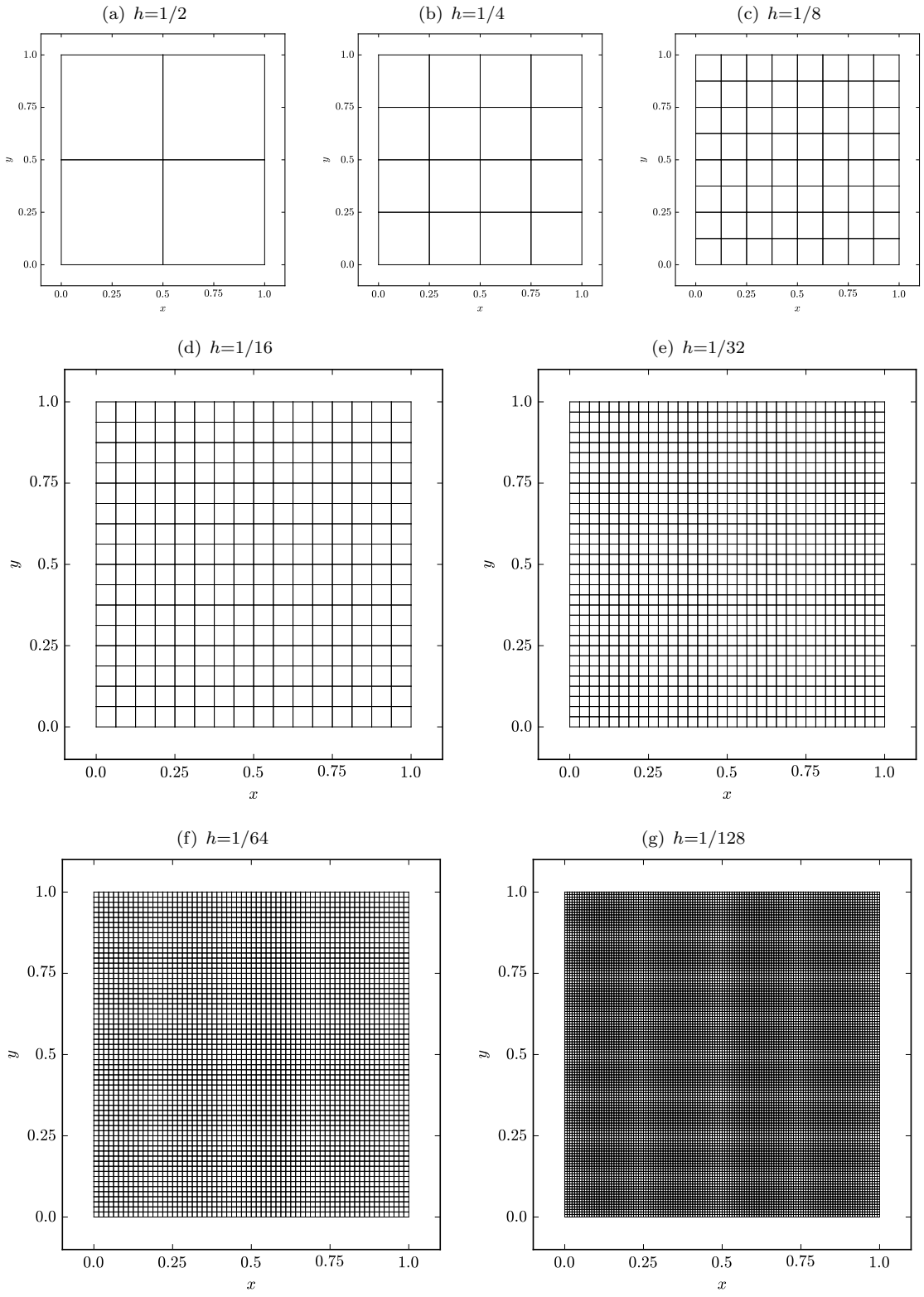


FIGURE A.1: Structured quadrilateral meshes for the 2D domain in section 4.1.1.

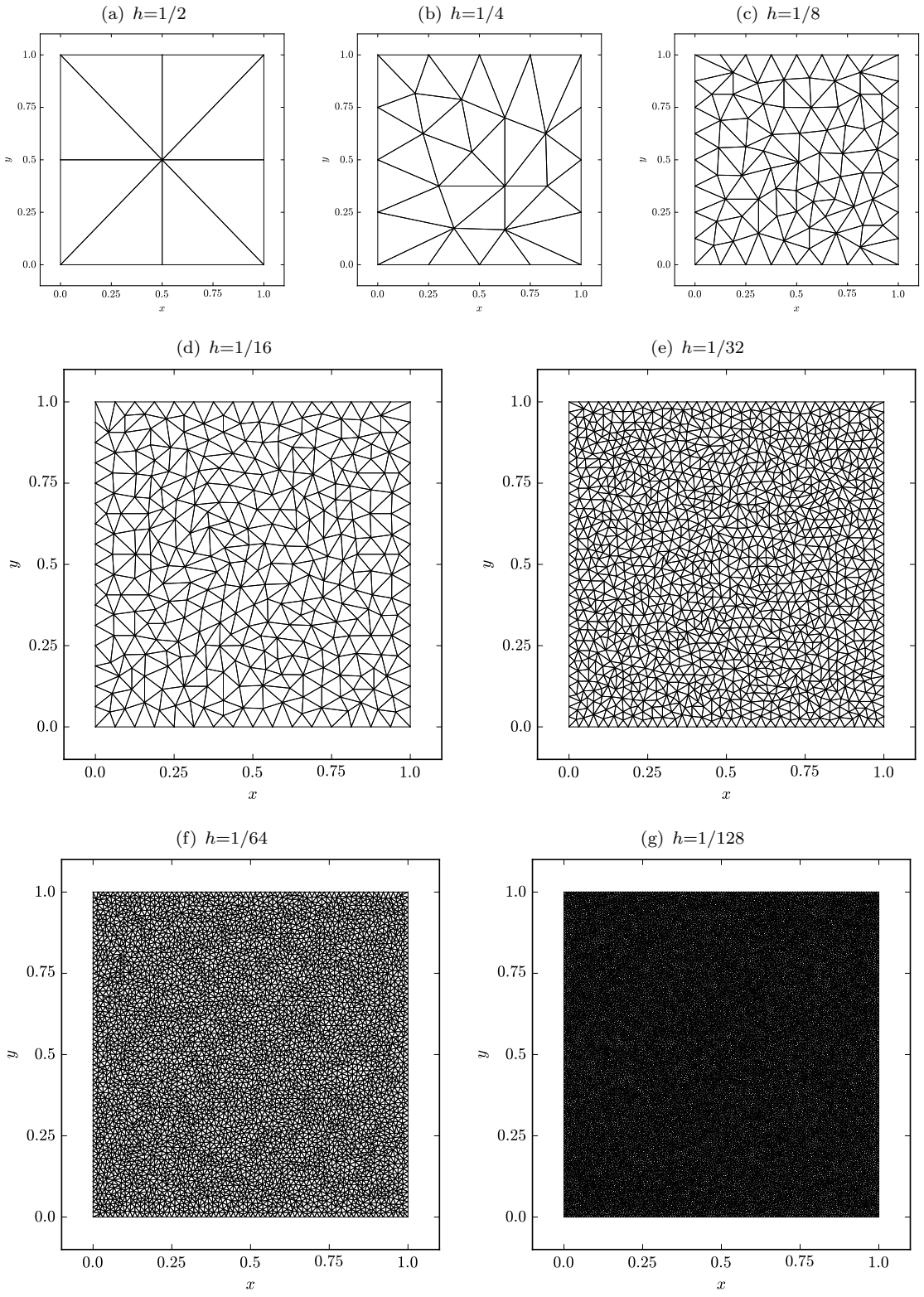


FIGURE A.2: Unstructured triangular meshes for the 2D domain in section 4.1.1.

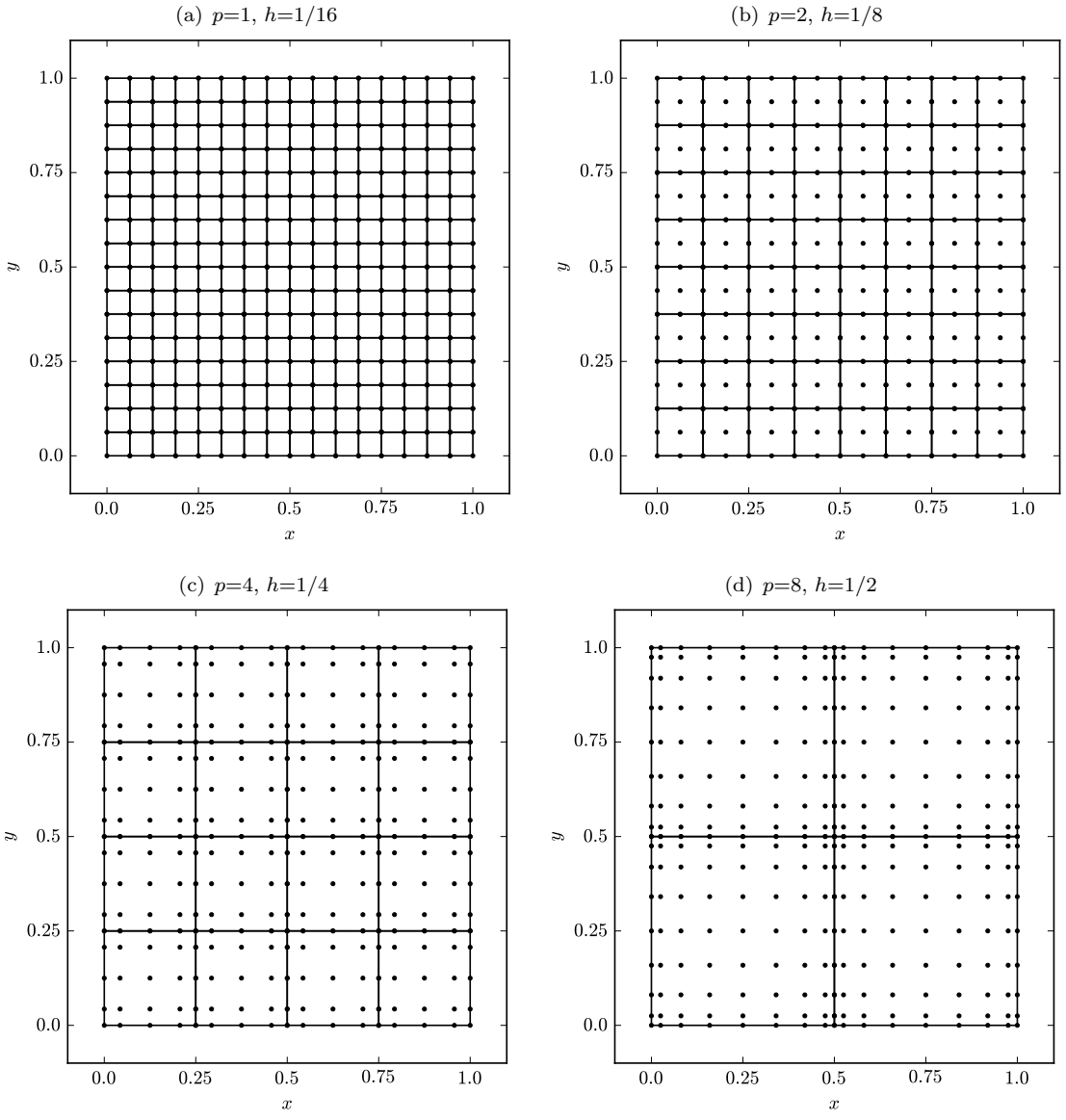


FIGURE A.3: Structured quadrilateral meshes for the 2D domain in section 4.1.2.

Appendix B: Individual performance results

This appendix contains the individual results of the performance analyses of the computations in section 4.1.1.

Figures B.1, B.2 and B.3 show the memory consumption of the different computations in 4.1.1 using structured quadrilateral, unstructured triangular ans structured hexahedral meshes respectively, depending on the polynomial degree p and on the mesh element size h.

Figures B.4, B.5 and B.6 show the time consumptions of the different computations in 4.1.1 using structured quadrilateral, unstructured triangular ans structured hexahedral meshes respectively, depending on the polynomial degree p and on the mesh element size h.

Figures B.7, B.8 and B.9 show the number of unknowns of the global system of the different computations in 4.1.1 using structured quadrilateral, unstructured triangular ans structured hexahedral meshes respectively, depending on the polynomial degree p and on the mesh element size h.

Figures B.10, B.11 and B.12 show the sparsity index of the global matrix of the different computations in 4.1.1 using structured quadrilateral, unstructured triangular ans structured hexahedral meshes respectively, depending on the polynomial degree p and on the mesh element size h.

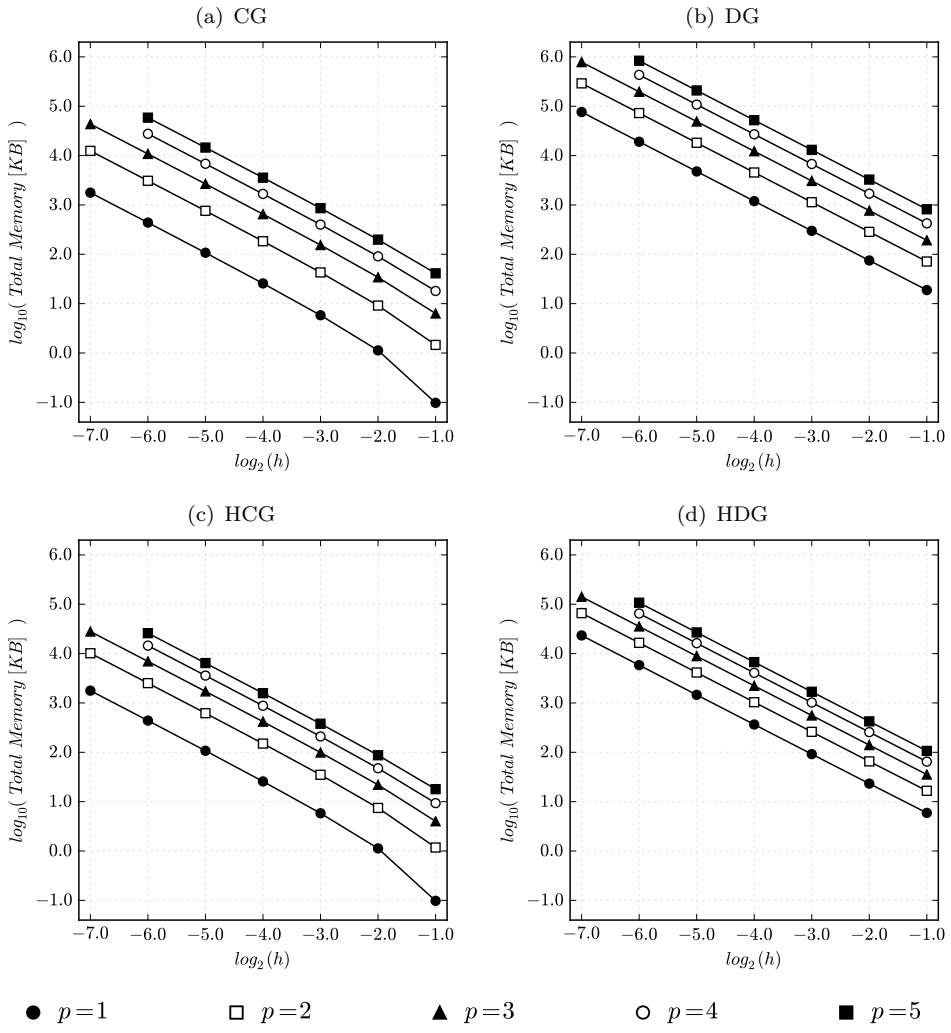


FIGURE B.1: Total memory consumption on structured quadrilateral meshes.

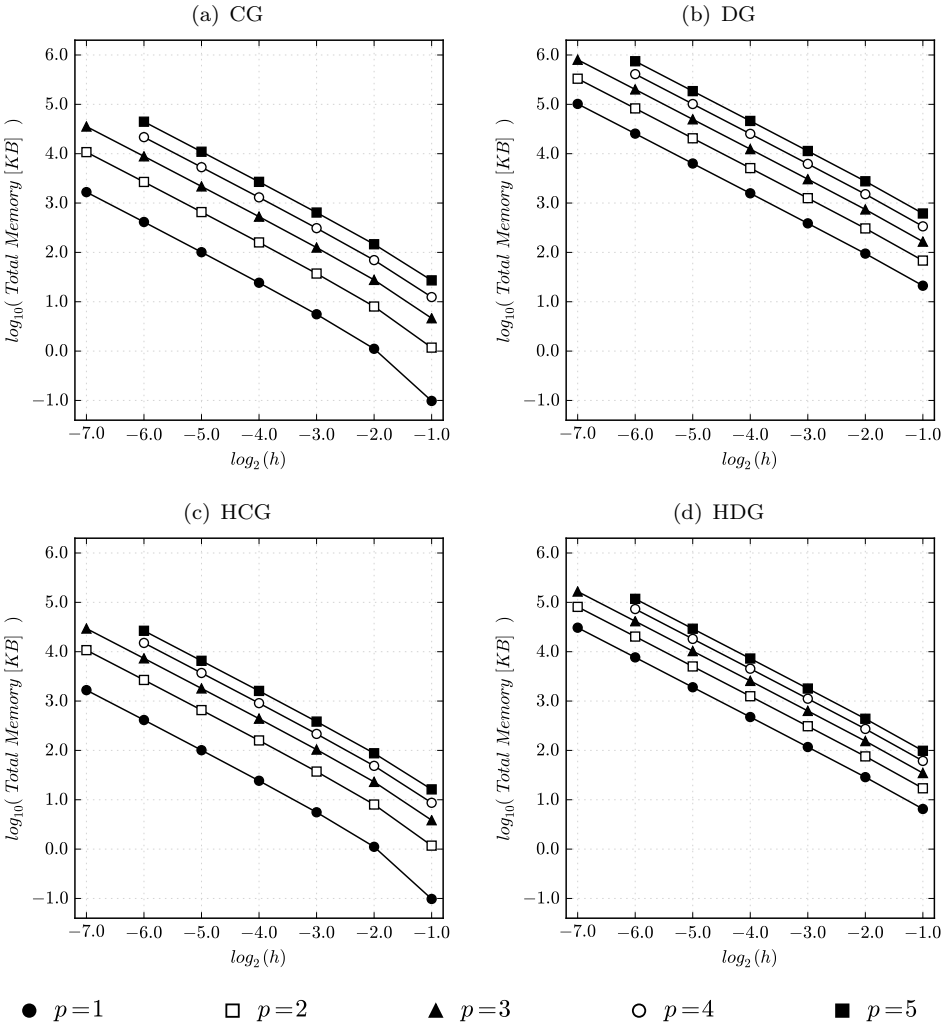


FIGURE B.2: Total memory consumption on unstructured triangular meshes.

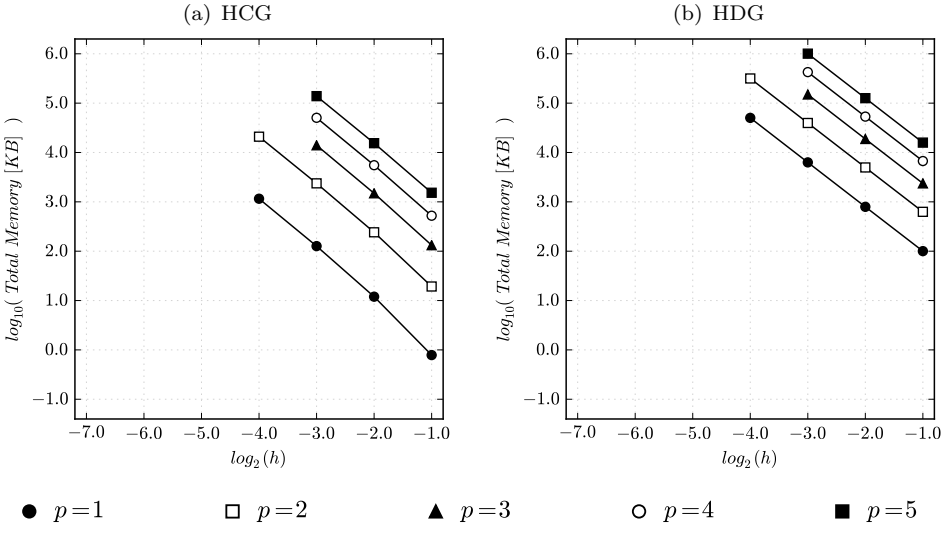
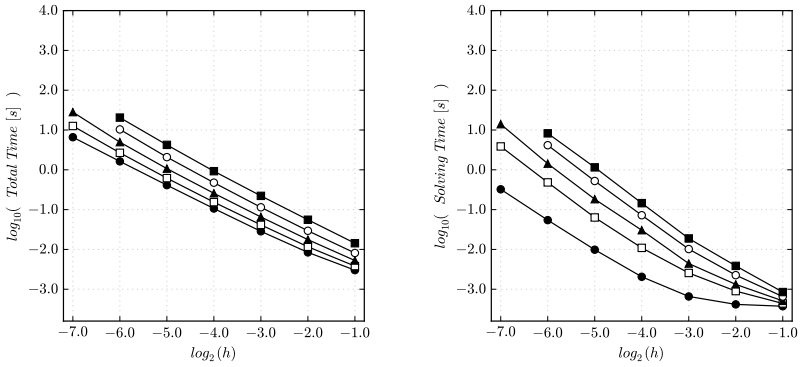
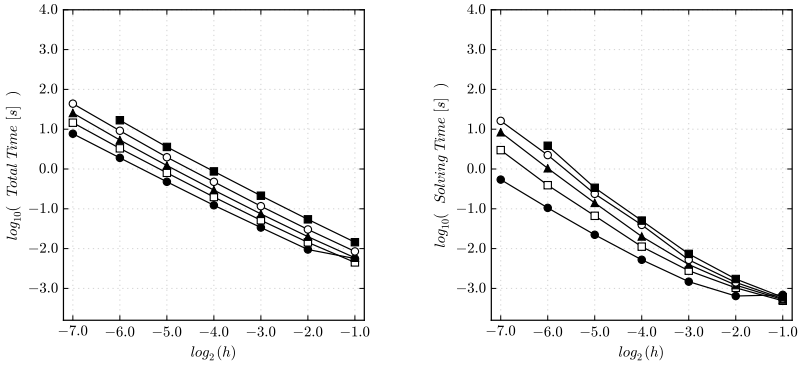


FIGURE B.3: Total memory consumption on structured hexahedral meshes.

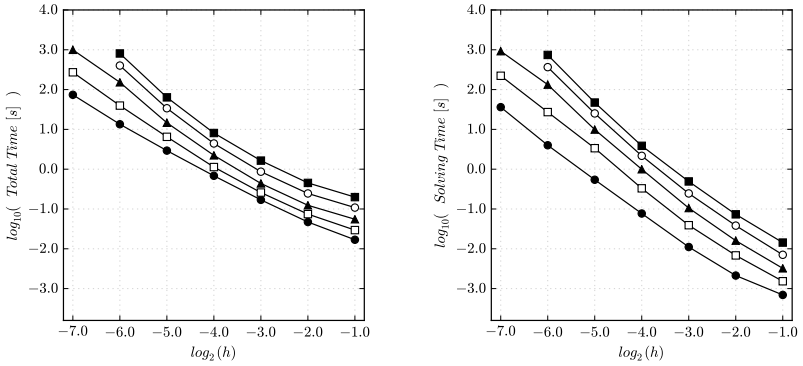
(a) Total and solving times for CG



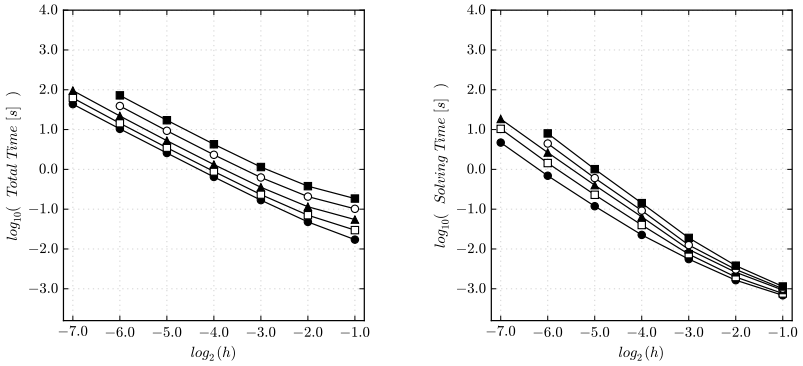
(b) Total and solving times for HCG



(c) Total and solving times for DG



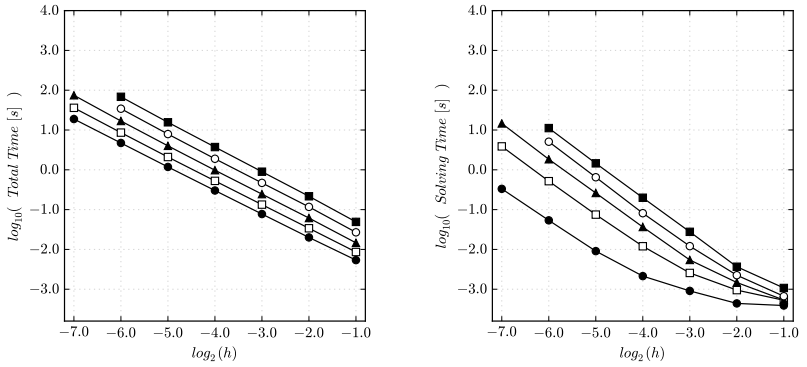
(d) Total and solving times for HDG



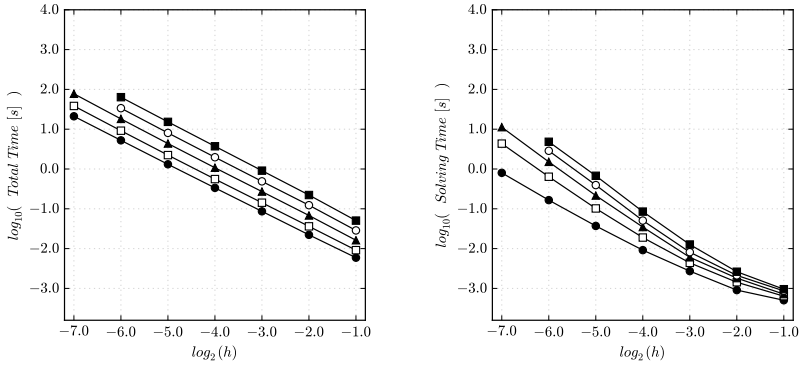
● $p=1$ □ $p=2$ ▲ $p=3$ ○ $p=4$ ■ $p=5$

FIGURE B.4: Time consumptions on structured quadrilateral meshes.

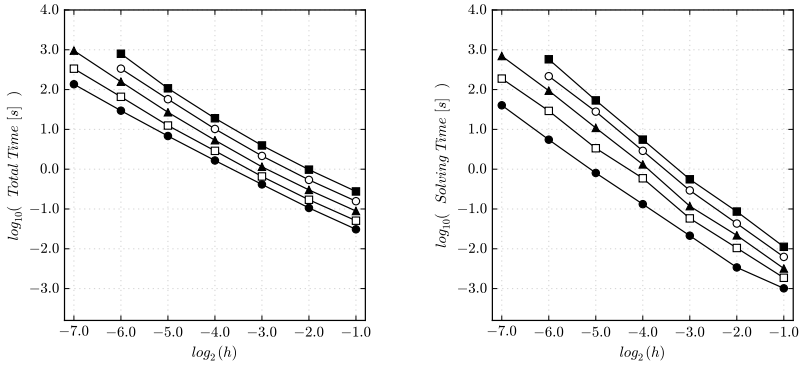
(a) Total and solving times for CG



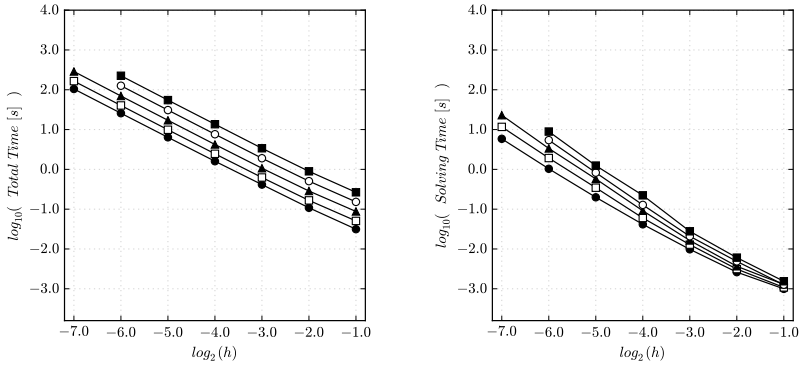
(b) Total and solving times for HCG



(c) Total and solving times for DG



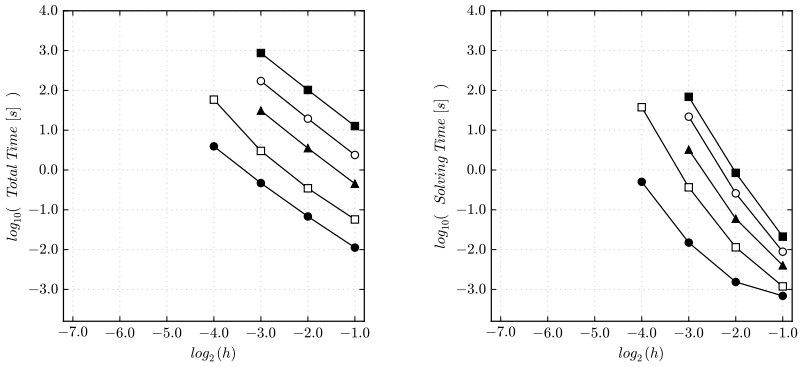
(d) Total and solving times for HDG



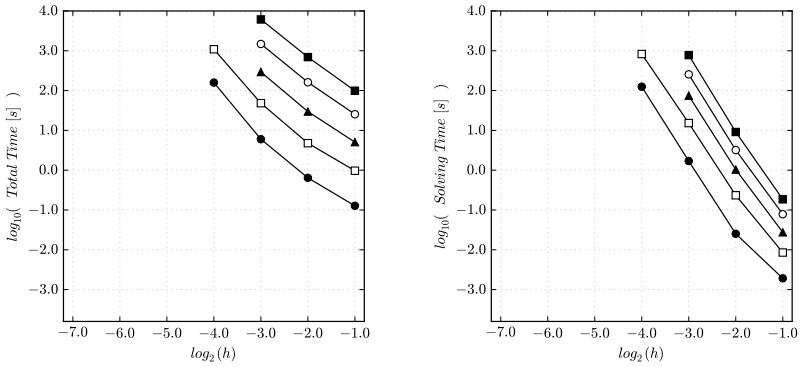
● $p=1$ □ $p=2$ ▲ $p=3$ ○ $p=4$ ■ $p=5$

FIGURE B.5: Time consumptions on unstructured triangular meshes.

(a) Total and solving times for HCG



(b) Total and solving times for HDG



● $p=1$ □ $p=2$ ▲ $p=3$ ○ $p=4$ ■ $p=5$

FIGURE B.6: Time consumptions on structured hexahedral meshes.

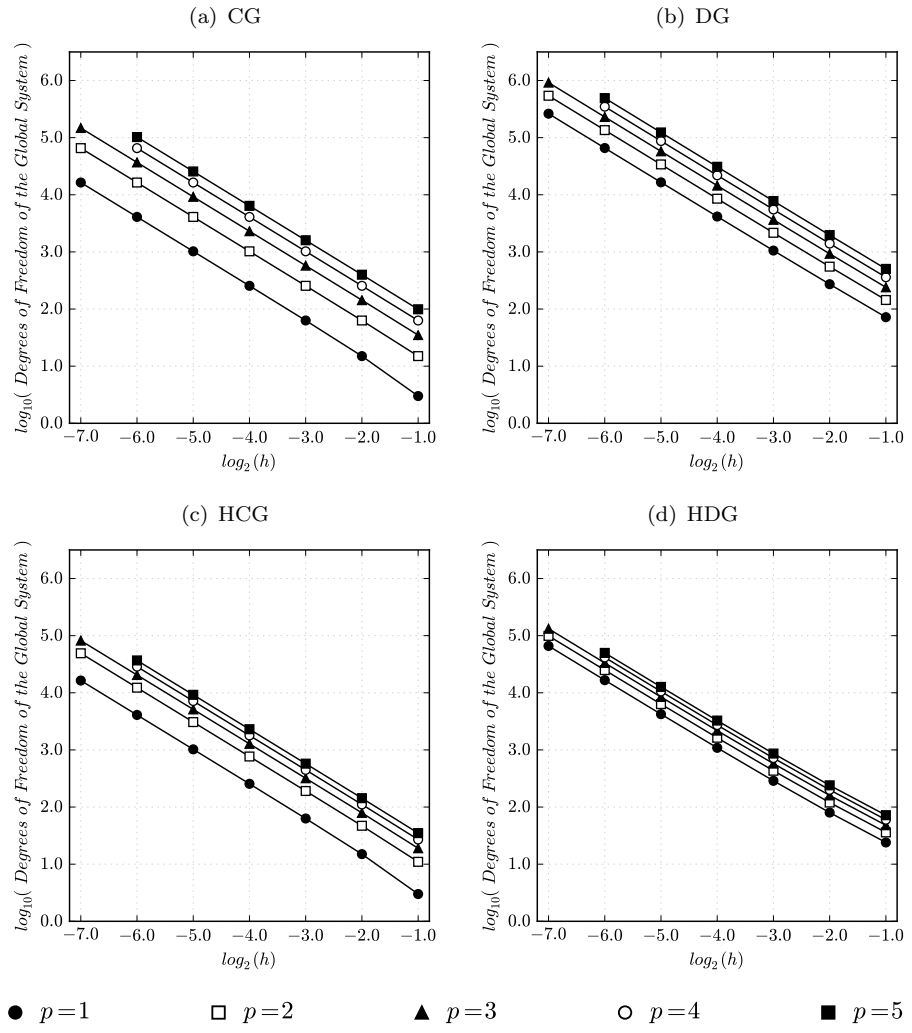


FIGURE B.7: Number of unknowns on structured quadrilateral meshes.

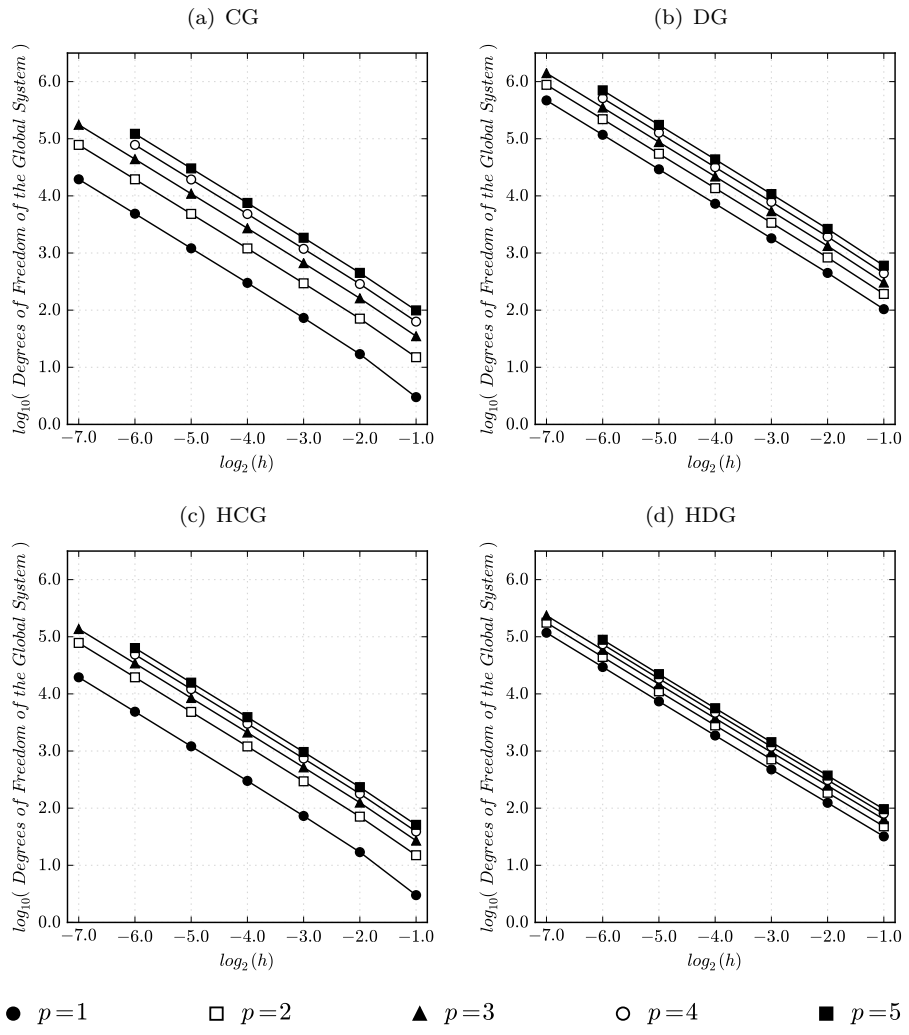


FIGURE B.8: Number of unknowns on unstructured triangular meshes.

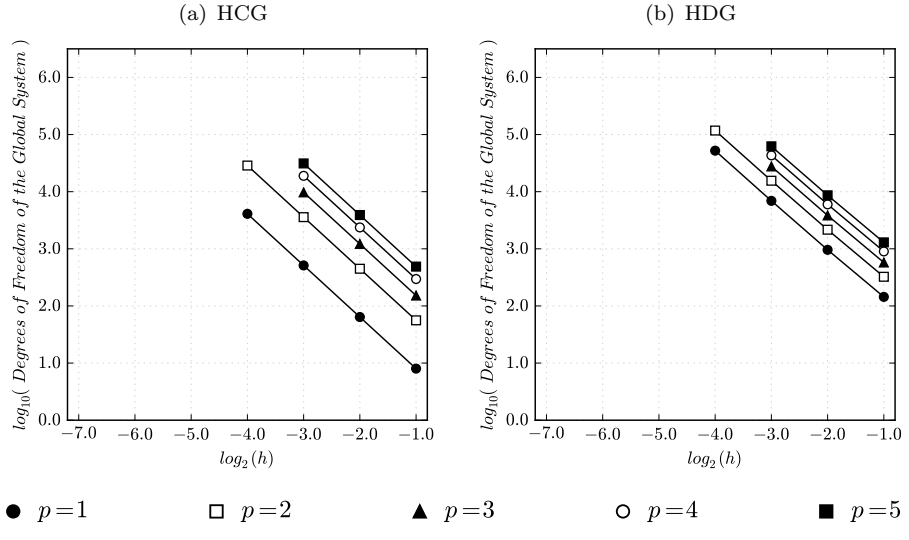


FIGURE B.9: Number of unknowns on structured hexahedral meshes.

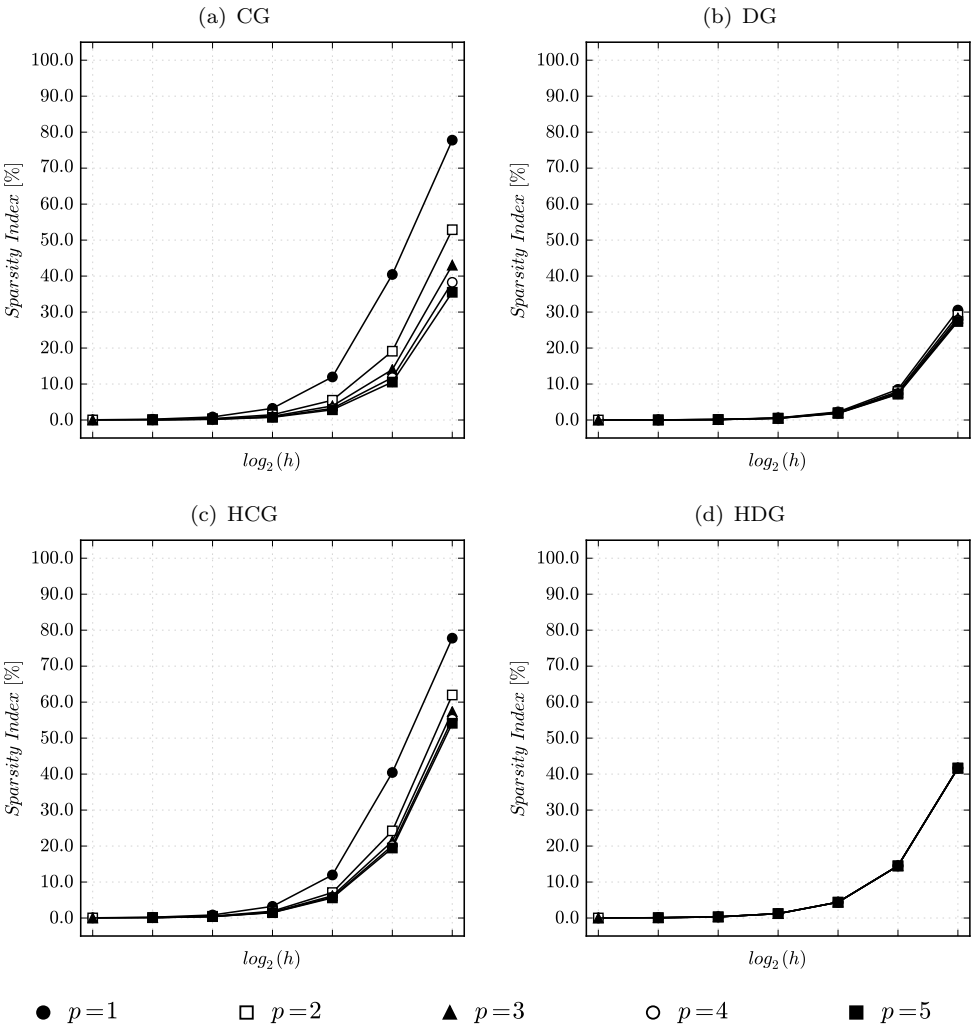


FIGURE B.10: Sparsity index of the global system on structured quadrilateral meshes.

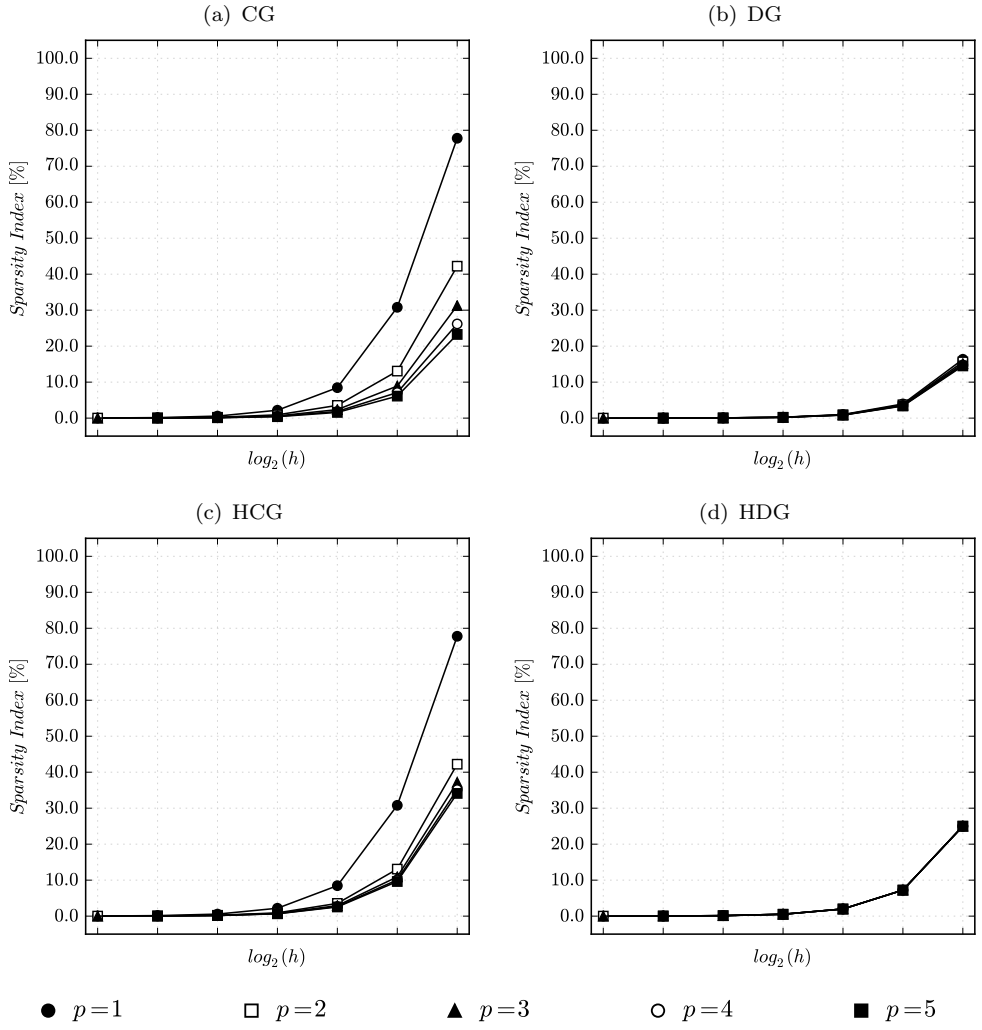


FIGURE B.11: Sparsity index of the global system on unstructured triangular meshes.

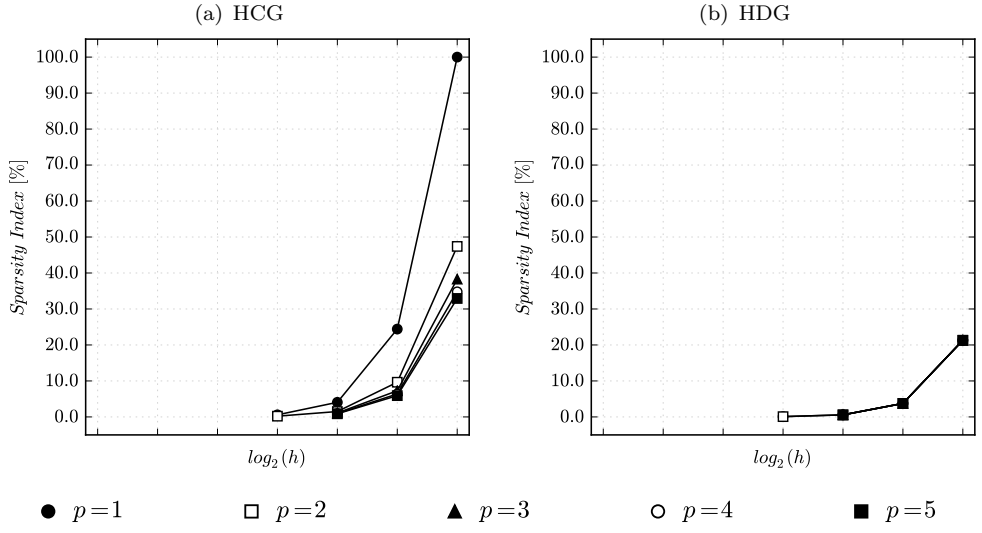


FIGURE B.12: Sparsity index of the global system on structured hexahedral meshes.

Bibliography

[1] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems. *SIAM J. Numer. Anal.*, 39 (5):1749–1779, May 2001. ISSN 0036-1429. URL http://dx.doi.org/10.1137/S0036142901384162.

[2] F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Methods*. Springer-Verlag New York, Inc., New York, NY, USA, 1991. ISBN 0-387-97582-9. URL http://dx.doi.org/10.1007/978-1-4612-3172-1.

[3] F. Brezzi, J. Douglas, and L. D. Marini. Two families of mixed finite elements for second order elliptic problems. *Numerische Mathematik*, 47(2):217–235. ISSN 0945-3245. URL http://dx.doi.org/10.1007/BF01389710.

[4] B. Cockburn and J. Gopalakrishnan. A characterization of hybridized mixed methods for second order elliptic problems. *SIAM J. Numer. Anal*, pages 283–301, 2004. URL http://dx.doi.org/10.1137/S0036142902417893.

[5] B. Cockburn and C. Shu. The Local Discontinuous Galerkin Method for Time-Dependent Convection-Diffusion Systems. *SIAM Journal on Numerical Analysis*, 35(6):2440–2463, 1998. URL http://dx.doi.org/10.1137/S0036142997316712.

[6] B. Cockburn, B. Dong, and J. Guzmán. A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems. *Math. Comp.*, 77(264):1887–1916, 2008. URL http://dx.doi.org/10.1090/S0025-5718-08-02123-6.

[7] B. Cockburn, J. Gopalakrishnan, and R. Lazarov. Unified Hybridization of Discontinuous Galerkin, Mixed, and Continuous Galerkin Methods for Second Order Elliptic Problems. *SIAM Journal on Numerical Analysis*, 47(2):1319–1365, 2009. URL http://dx.doi.org/10.1137/070706616.

[8] B. Cockburn, J. Guzmán, and H. Wang. Superconvergent discontinuous Galerkin methods for second-order elliptic problems. *Math. Comp.*, 78(265):1–24, 2009. URL http://dx.doi.org/10.1090/S0025-5718-08-02146-7.

[9] J. Douglas and T. Dupont. *Computing Methods in Applied Sciences: Second International Symposium December 15–19,1975*, chapter Interior Penalty Procedures for Elliptic and Parabolic Galerkin Methods, pages 207–216. Springer Berlin Heidelberg, Berlin, Heidelberg, 1976. ISBN 978-3-540-37550-0. URL http://dx.doi.org/10.1007/BFb0120591.

[10] T. L. Forti, A. M. Farias, P. R. Devloo, and S. M. Gomes. A comparative numerical study of different finite element formulations for 2D model elliptic problems: Continuous and discontinuous Galerkin, mixed and hybrid methods . *Finite Elements in Analysis and Design*, 115:9 – 20, 2016. ISSN 0168-874X. URL http://dx.doi.org/10.1016/j.finel.2016.02.009.

[11] A. Gargallo-Peiró, X. Roca, J. Peraire, and J. Sarrate. Optimization of a regularized distortion measure to generate curved high-order unstructured tetrahedral meshes. *International Journal for Numerical Methods in Engineering*, 103(5):342–363, 2015. ISSN 1097-0207. URL http://dx.doi.org/10.1002/nme.4888.

[12] A. Gargallo-Peiró, X. Roca, J. Peraire, and J. Sarrate. A distortion measure to validate and generate curved high-order meshes on CAD surfaces with independence of parameterization. *International Journal for Numerical Methods in Engineering*, 106(13):1100–1130, 2016. ISSN 1097-0207. URL http://dx.doi.org/10.1002/nme.5162.

[13] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[14] G. Karniadakis and S. J. Sherwin. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press on Demand, 2005.

[15] R. M. Kirby, S. J. Sherwin, and B. Cockburn. To CG or to HDG: A Comparative Study. *Journal of Scientific Computing*, 51(1):183–212, 2011. ISSN 1573-7691. URL http://dx.doi.org/10.1007/s10915-011-9501-7.

[16] Kitware Inc. and Los Alamos National Laboratory. ParaView: Open-source, multi-platform data analysis and visualization application, 2000. URL http://www.paraview.org.

[17] LaCàN. EZ4U: Mesh generation environment, 2001. URL https://www.lacan.upc.edu/ez4u.htm.

[18] LaCàN. Cluster Clonetroop, 2011. URL https://www.lacan.upc.edu.

[19] K. Nakajima. *Parallel Multistage Preconditioners Based on a Hierarchical Graph Decomposition for SMP Cluster Architectures with a Hybrid Parallel Programming*

Model, pages 384–395. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-75444-2. URL http://dx.doi.org/10.1007/978-3-540-75444-2_39.

[20] N. Nguyen, J. Peraire, and B. Cockburn. An implicit high-order hybridizable discontinuous Galerkin method for linear convection-diffusion equations. *Journal of Computational Physics*, 228(9):3232 – 3254, 2009. ISSN 0021-9991. URL http://dx.doi.org/10.1016/j.jcp.2009.01.030.

[21] M. Paz and W. Leigh. *Integrated Matrix Analysis of Structures: Theory and Computation*, chapter Static Condensation and Substructuring, pages 239–260. Springer US, Boston, MA, 2001. ISBN 978-1-4615-1611-8. URL http://dx.doi.org/10.1007/978-1-4615-1611-8_8.

[22] Python Software Foundation. Python: Open-source, high-level, interpreted and object-oriented programming language, 1989. URL https://www.python.org.

[23] P. A. Raviart and J. Thomas. *Mathematical Aspects of Finite Element Methods: Proceedings of the Conference Held in Rome, December 10–12, 1975*, chapter A mixed finite element method for 2-nd order elliptic problems, pages 292–315. Springer Berlin Heidelberg, Berlin, Heidelberg, 1977. ISBN 978-3-540-37158-8. URL http://dx.doi.org/10.1007/BFb0064470.

[24] W. H. Reed and T. R. Hill. *Triangular mesh methods for the neutron transport equation*. Oct 1973. URL http://www.osti.gov/scitech/servlets/purl/4491151.

[25] E. Ruiz-Gironés, X. Roca, and J. Sarrate. High-order mesh curving by distortion minimization with boundary nodes free to slide on a 3D CAD representation. *Computer-Aided Design*, 72:52 – 64, 2016. ISSN 0010-4485. URL http://dx.doi.org/10.1016/j.cad.2015.06.011. 23rd International Meshing Roundtable Special Issue: Advances in Mesh Generation.

[26] R. Stenberg. Postprocessing schemes for some mixed finite elements. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 25(1):151–167, 1991. URL http://eudml.org/doc/193618.

[27] T. Vejchodský and P. Šolín. Static condensation, partial orthogonalization of basis functions, and ILU preconditioning in the hp-FEM. *Journal of Computational and Applied Mathematics*, 218(1):192 – 200, 2008. ISSN 0377-0427. URL http://dx.doi.org/10.1016/j.cam.2007.04.044.

[28] The Scipy Community. Data types, 2008. URL http://docs.scipy.org/doc/numpy-1.10.1/user/basics.types.html.

[29] The Scipy Community. Sparse matrices, 2008. URL http://docs.scipy.org/doc/scipy/reference/sparse.html.

[30] E. L. Wilson. The static condensation algorithm. *International Journal for Numerical Methods in Engineering*, 8:198–203, 1974. URL http://dx.doi.org/10.1002/nme.1620080115.

[31] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. ISBN 0750663200.